ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL
DA TESE DEFENDIDA POR Orlando Maurício
Durán Acevedo E APROVADA PELA
COMISSÃO JULGADORA EM 15 3 93

ORIENTADOR

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE ENGENHARIA DE MATERIAIS

Dissertação apresentada à
Faculdade de Engenharia Mecânica
Como Requisito Parcial à Obtenção do
Título de Mestre em Engenharia Mecânica

SIMULADOR GRÁFICO PARA UM ROBÔ INDUSTRIAL

Autor :Orlando Mauricio Durán Acevedo
Orientador :Geraldo Nonato Telles OK

11/93

Março de 1993

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE ENGENHARIA DE MATERIAIS

A dissertação "Simulador Gráfico para um Robô Industrial", elaborada por Orlando Mauricio Durán Acevedo foi aprovada por:

Dr. Geraldo Nonato Telles
(Orientador)

Dr. Márcio Luis de Andrade Neto

Dr. Maurício Prattes Campos Filho

Campinas, 15 de março de 1993

Dedico meu esforço

A meus pais, Orlando e Pilar
A meus Irmaôs, Paulo e Pily

RESUMO

Nesta dissertação é especificado e implantado um protótipo de um sistema de simulação "Off-line" para um robô industrial do tipo SCARA[1] (IBM 7535).

Trata-se de um pacote de software composto de quatro módulos principais : Um interpretador da linguagem AML[2], um gerenciador de objetos e "Lay-Outs", um simulador gráfico com seis possibilidades de visualização e um detetor automático de colisões baseado na construção de Mapas de Configuração.

O simulador foi implantado num microcomputador sob o Sistema Operacional MS-DOS.

ABSTRACT

This dissertation presents the specification and implementation of a SCARA[1] type Industrial Robot Off-line Simulation System Prototype (IBM 7535).

The system is composed by four main modules : An AML[2] language Interpreter, a lay-outs managment system, a graphic simulator with six options to watching the manipulator's actions and a Configuration Map-Based Automatic collisions detector.

This system was implemented under the MS-DOS operating System.

---

[1] Selective Compliance Assembly Robot Arm

[2] A Manufacturing Language

AGRADECIMENTOS

A Nelson Durán, meu tio,

Por ter me recibido no seu lar e me dar a possibilidade de levar a cabo este trabalho. Suas palavras sempre foram orientadoras e motivantes, e através delas colocou em mim a semente do interesse pela pesquisa. Muito Obrigado.

Ao Professor Geraldo Nonato Telles, pela oportunidade de ingresso na Pós-graduação e por ter me confiado a responsabilidade de executar este projeto.

À Sheila pela amizade, carinho, companhia e alegrias que ela sabe dar.

Aos Colegas do DEMA e do DEF, pela amizade, companherismo e, por sobretudo, por ter me recibido como um mais deles.

Ao CNPq,Conselho Nacional de Desenvolvimento Científico e Tecnológico pelo apoio financeiro.

INDICE

ABREVIAÇÕES

AML	:	A Manufacturing Language
ASCII	:	American Standart Code for Information Interchange.
CAD	:	Computer-Aided Design (Projeto Assistido por Computador)
CAM	:	Computer-Aided Manufacturing (Manufatura Assistida por Computador)
CIM	:	Computer Integrated Manufacturing (Manufatura Integrada por Computador)
IGES	:	International Graphic Exchange Standart Padrão Internacional para Transferência Gráfica
IRDATA	:	Industrial Robot Data
SCARA	:	Selective Compliance Assembly Robot Arm

LISTA DE FIGURAS

CAPITULO 1.

CAPITULO 3.

CAPITULO 4.

LISTA DE TABELAS

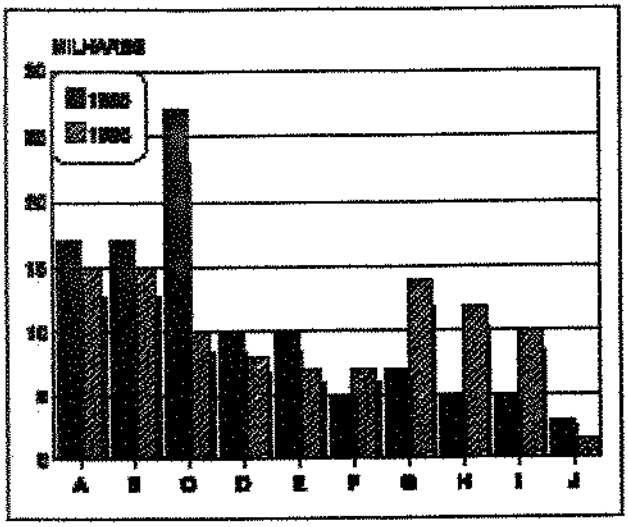
Capítulo 2

Capítulo 3

Capítulo 4

1.INTRODUÇÃO

No começo os robôs industriais foram dedicados a tarefas simples e repetitivas. Estas tarefas não requeriam alta precisão nem flexibilidade. Porém, atualmente, como pode ser observado na figura 1.1. a partir de estudos feitos nos Estados Unidos, a tendência indica que os robôs industriais incrementarão a sua participação em atividades onde é requerida maior capacidade e sofisticação. Uma atividade que verá incrementada substancialmente a sua incorporação ao campo da robótica industrial é a montagem, especificamente a montagem de componentes e conjuntos eletrônicos.



A Alimentação de Máq.Ferr.
B Transferência de Material
C Soldagem a Ponto
D Soldagem a Arco
E Pintura/Revestimento
F Processamento*
G Mont.Eletrônica
H Outras Montagens
I Inspeção
J Outros
* Inclui aplic. de manipulação de ferramental.

Fig. 1.1. Distribuição de robôs por aplicação nos E.U.A.

Se as tendências confirmarem e as pesquisas em curso forem convertidas em tecnologia, a população média de robôs industriais estará destinada a sofrer um importante incremento.

Observe, por exemplo, os dois gráficos seguintes.

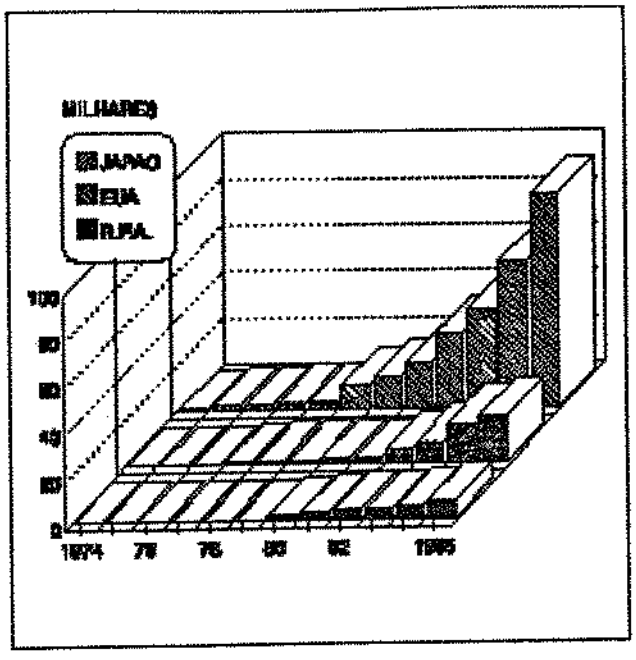


Fig. 1.2. População de robôs industriais em três países industrializados

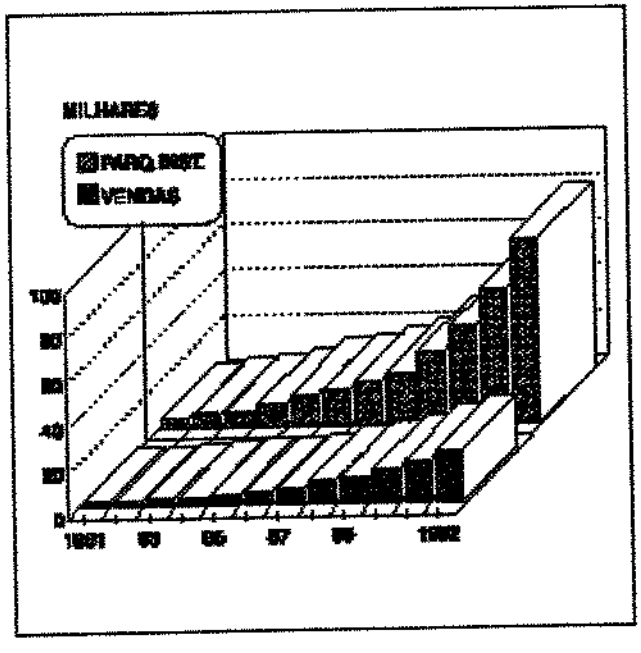


Fig. 1.3. Vendas e parque instalado de robôs industriais nos E.U.A.

No gráfico da figura 1.2. é feita uma comparação entre os tres países mais industrializados do mundo. Todos eles mostraram um crescimento contínuo na sua população de

robôs industriais. Este incremento pode ser observado, mesmo desde a incorporação à atividade produtiva no começo dos anos 70.

Isto pode ser confirmado na figura 1.3, onde com base em uma projeção feita nos Estados Unidos **[GROO-89]** pode-se ver que tanto as vendas, quanto o parque instalado de robôs industriais nesse pais terão um crescimento exponencial em curto prazo.

Este incremento deve-se principalmente aos seguintes fatores :

○ Massificação da tecnologia robótica e o seu alto potencial para as mais diversas aplicações.

○ Simplificação dessa tecnologia, o que tornará os robôs mais "amistosos", mais fáceis de interfacear com outros equipamentos e sistemas (hardware e software) e mais fáceis de instalar.

○ Baixo custo de produção e, portanto, baixos preços dos robôs industriais.

○ Expansão do mercado para pequenas e médias empresas, além das grandes empresas, que continuarão usando instalações flexíveis.

Por outro lado, existem outros dois fatos significativos. Primeiro, o mercado ao expandir-se, provocará uma alta competição entre fabricantes e vendedores deste tipo de sistemas. Esta alta competição obrigará aos fabricantes cada vez a oferecer mais e mais ferramentas e accessórios relacionados a esta tecnologia, por exemplo sistemas CAD para programação, novos protocolos de comunicação, etc.

Isto implicará que o custo do manipulador em si, cada vez corresponda a uma parcela menor dentro do custo total de implantação de um sistema robotizado. Na atualidade o custo do robô compreende mais ou menos o 50% do custo total (figura 1.4.).

Porém, a atenção especial que está se dando aos sistemas de controle, software e atividades de suporte tenderão a diminuir cada vez mais essa porcentagem. Este fato coincide com a tendência geral da automação industrial, onde a importância do hardware já foi ultrapassada pela importância do software e da organização. **[OTA-84]**.

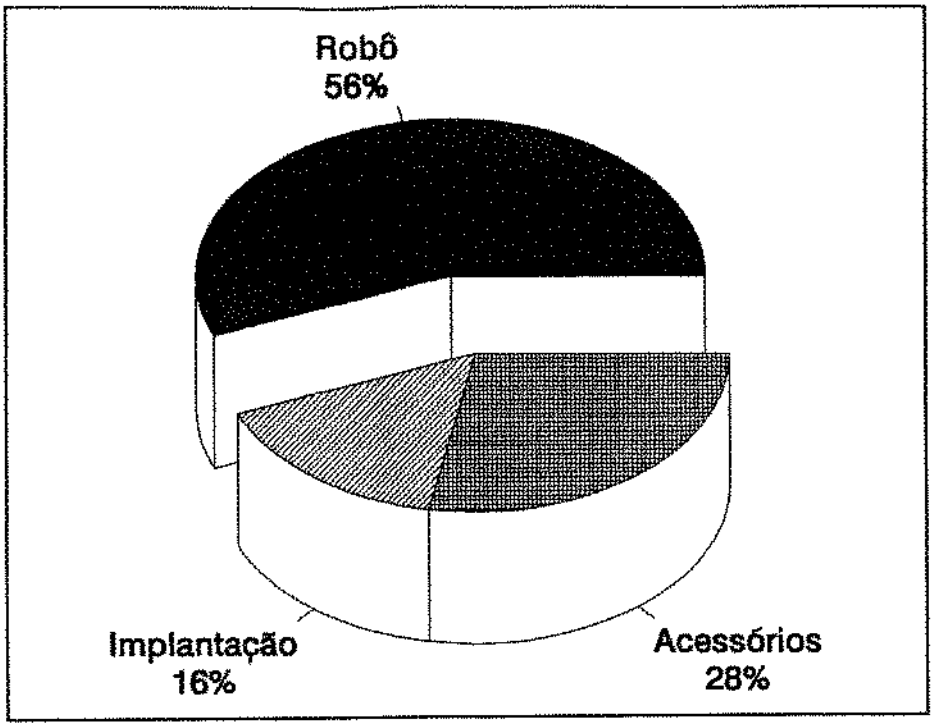


Fig. 1.4. Estrutura de custos de um sistema robotizado

Analizando-se o gráfico da figura 1.5. vê-se que, apesar dos robôs industriais de sequencia fixa e variável e os programados por "play-back" ocuparem os primeiros lugares no parque instalado, estes estão destinados a ser substituidos pelos robôs comandados numericamente e pelos robôs "inteligentes". Blumenthal [BLUM-90] destacou que a proporção de robôs de sequência fixa e variável, no parque instalado no Japão, diminuiu de 77%, em 1980, para 24% em 1987. Entretanto, os robôs controlados numericamente, aumentaram de 6% para 30% no mesmo periodo de tempo.

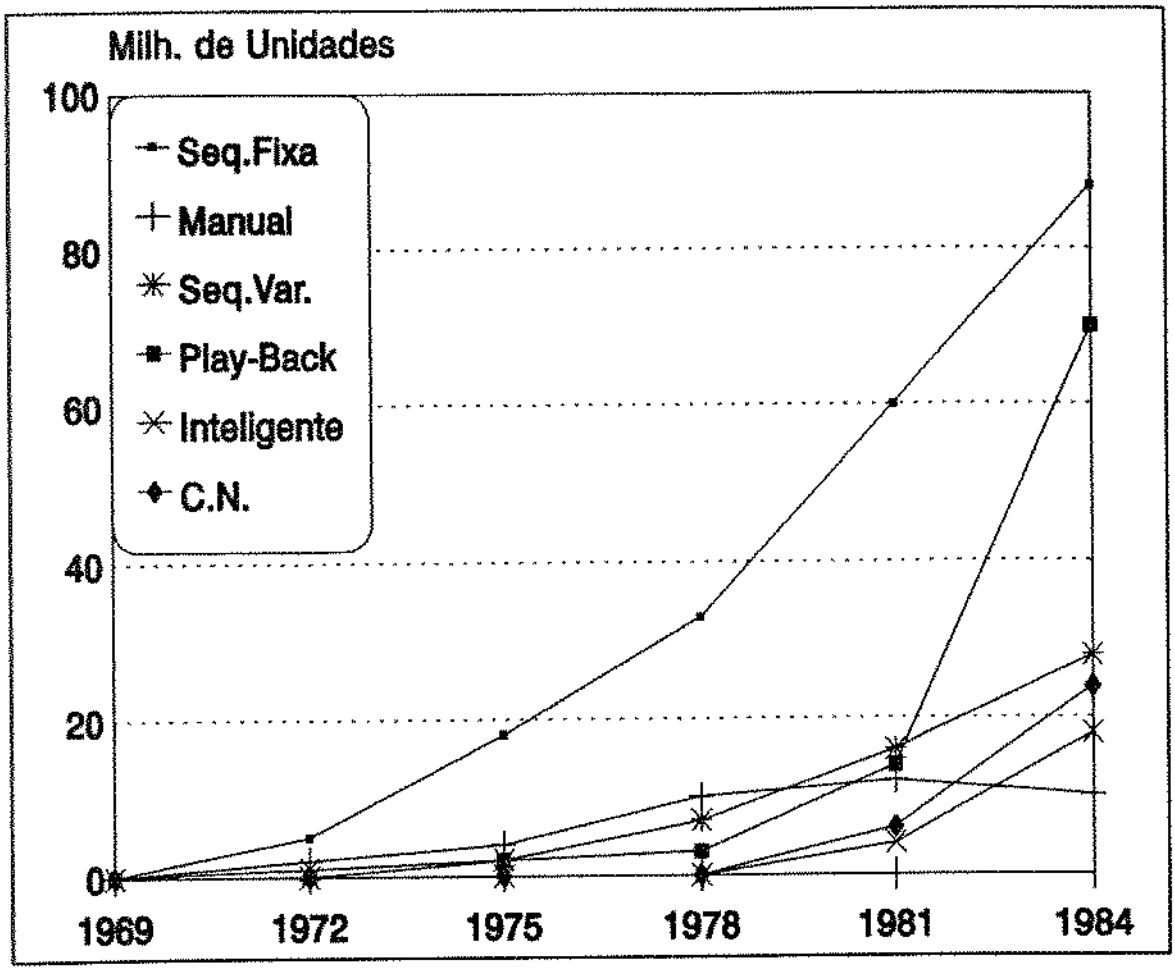


Fig. 1.5. População de robôs industriais segundo método de programação

Isto implica em um incremento das necessidades de sistemas de programação "off-line" e sistemas de suporte ao programador, de maneira a simplificar estas tarefas e viabilizá-las para o ambiente industrial.

A seguir são analisadas mais detalhadamente as atividades relacionadas com a programação "off-line" de robôs industriais.

1.1. Justificativa

Tem-se observado que a etapa de depuração e teste de um programa de aplicação para robôs demora até mais do dobro do tempo requerido no desenvolvimento deste programa **[BUCK-85]**. As principais causas dessa demora são as seguintes :

○ A justificativa econômica na implantação de sistemas robotizados em aplicações industriais está prioritariamente baseada no tempo de ciclo **[NWAG-91]**, ou seja, quanto mais rápido for realizada a tarefa em questão, mais aconselhável será a implantação deste tipo de aplicação. Portanto, os programadores gastam grande parte do seu tempo no esforco de otimizar programas que viabilizem a redução nos tempos de fabricação.

○ Durante a etapa de depuração de um programa, ocorre uma série de problemas relacionados com a precisão dos pontos definidos **[EDKI-85]**. Alguns destes problemas são solucionados ajustando os pontos. Isto pode ser feito através de uma representação simbólica (parâmetros e comandos) ou através da definição explícita das posições, orientações e dimensões provenientes do próprio ambiente e obtidas através de algum dispositivo de medição **[HALM-88]**.

Outros problemas podem ser resolvidos mediante a diminuição da velocidade dos movimentos do manipulador, e inclusive alguns são solucionados através de mudanças na estrutura algorítmica do programa.

○ É muito difícil para o programador antecipar-se a todos os problemas que podem acontecer numa aplicação complexa; assim a adição de comandos ("BreakPoints"), no corpo do programa para manejar a ocorrência de erros não previstos é uma prática muito comum nesta etapa de depuração. O objetivo é adicionar ao sistema robustez e a capacidade de reação apropriada frente a possíveis condições de erro, tanto internas como externas ao sistema **[SMIT-85]**.

Portanto, considerando a tendência da realidade industrial de diminuir os tempos de preparação entre diferentes produtos nas linhas de produção, torna-se importantíssimo otimizar os tempos de programação e teste dos equipamentos robotizados; essa realidade é um desafio e uma grande chance de sucesso para o uso de sistemas de programacao "off-line".

1.2. Objetivos

Usualmente a programação "off-line" combina o uso da simulação gráfica para produzir e analisar um plano desejado de trajetórias e/ou o programa de aplicação **[STOB-85]**. Assim, o movimento do manipulador, e ocasionalmente, a interação com outros equipamentos são mostrados na tela de um computador, permitindo suscessivas depurações do programa, diminuindo sensivelmente o tempo requerido pelas etapas de depuração e "try-out" **[FELD-91]**.

Este tipo de ferramentas faz com que os engenheiros reduzam em cerca de 30% os tempos de planejamento e programação de sistemas robotizados **[CHAN-88]**.

Diversos sistemas têm sido criados, com diferentes graus de sofisticação, existindo desde sistemas que representam esquematicamente o manipulador e os seus movimentos **[WECK-87] [LANG-87]**, até versões bem mais complexas que usam recursos CAD em 3 dimensões e técnicas de modelamento de sólidos permitindo inclusive a detecção automática de colisões **[BONN-87] [STOB-87] [LEE-90]**. Esta última geração de sistemas, infelizmente deriva em altos custos de investimento em equipamento computacional.

Dentro deste contexto, e com o objetivo específico de facilitar a programação "off-line" e os testes posteriores dos programas criados para um robô industrial, desenvolveu-se um sistema que auxilia ao programador na definição e visualização das trajetórias definidas por um programa na linguagem AML. Como plataforma básica de teste do sistema utilizou-se um robô tipo SCARA da IBM modelo 7535.

Um dos objetivos propostos é desenvolver-se um software de apoio de baixo custo e simplicidade operacional e, considerando o fato que os sistemas de simulação gráfica baseados

em PCs normalmente constituem menos da metade do preço dos sistemas baseados em arquiteturas mais complexas ("workstations" e " mainframes") [KARA-91], optou-se por microcomputadores da geração XT/AT (compatíveis com IBM), devido principalmente a seu baixo custo e à sua alta difusão e presença nas indústrias brasileiras e no mundo todo.

À nível de "software" o objetivo é criar um sistema que tenha uma interface Homem-Máquina bastante amigável, além de possuir uma estrutura modular que permita o seu fácil crescimento e modificação.

2. LINGUAGENS DE PROGRAMAÇÃO

O incremento da implantação e uso de robôs industriais na fabricação e montagem deve-se principalmente ao aumento da flexibilidade e da produtividade proporcionados por esses equipamentos, provocado principalmente pelo advento de novos métodos de programação.

Estes métodos foram diferenciados por Storr **[STOR-87]** em dois grandes grupos : Métodos Diretos e Métodos Indiretos.

2.1 Métodos Diretos :

Os métodos diretos são aqueles que usam o manipulador diretamente como ferramenta de programação e armazenamento dos programas. Por sua vez estes métodos estão divididos em tres sub-classes :

- Manual
- "Play-back"
- "Teach-in".

A programação manual é levada a cabo com a ajuda de um teclado ou "plug-board" mediante o qual o operador dirige diretamente os movimentos do manipulador. Na realidade este método não é nada mais do que a prolongação das capacidades humanas através de um dispositivo mecânico, que apenas obedece as ordens transmitidas pelo operador por meios eletromecânicos **[BEJC-85]**.

O método "play-back" consiste em conduzir manualmente o manipulador, ensinando-lhe a tarefa que haverá de realizar. Os programas feitos sob esse método são simples sequências de dados, não havendo normalmente a possibilidade de técnicas mais estruturadas de programação

como o uso de "loops", desvios condicionais, nem a sincronização com outros dispositivos existentes na aplicação [DEIS-85].

No método "teach-in" são armazenados os pontos indicados pelo programador através de um botão ou outro dispositivo sinalizador. As interpolações entre estes pontos são realizadas pelo próprio controlador do manipulador, distintamente do método "play-back" onde são memorizadas as trajetórias completas [DEIS-85].

A programação direta está sendo dia a dia substituida pela programação Indireta ou "off-line", já que esta última técnica não precisa acessar o equipamento para realizar a tarefa de programação, tirando-o da atividade produtiva. Outra consideração favorável ao método "off-line" de programação, é a questão de seguranca das pessoas envolvidas com o equipamento.

Isto é, existe uma diversidade de aplicações onde os robôs industriais operam em ambientes altamente perigosos e hostís para os seres humanos, e a participação direta do homem é limitada ou impossível [SALV-85] [MUNS-85]. Assim, a única maneira de se programar esses equipamentos é através da definição de tarefas a longa distância ou "off-line".

2.2. Programação Indireta

Na programação indireta a tarefa de programação é feita sem o robô industrial, em geral com o uso de uma linguagem orientada, desde um terminal remoto, um microcomputador ou uma estação de trabalho.

Com este propósito têm sido desenvolvidas linguagens de programação textual de alto nível (específicas para ambiente industrial) de maneira a facilitar a programação de tarefas complexas de uma forma mais natural.

Estas linguagens têm sido classificadas em dois tipos : Linguagens de programação Explícita e Linguagens de programação Implícita.

2.2.1 Programação Explícita

Na programação explícita, é o programador que define as tarefas específicas através do uso de instruções textuais de uma linguagem dedicada, por exemplo : MOVE, GRASP, UP, etc.. Gruver realizou uma análise comparativa entre as linguagens desse tipo **[GRUV-84]**. Segundo ele, durante a fase inicial estas linguagens foram obtidas a partir da extensão das linguagens de programação de propósito múltiplo. Foi assim que em 1974 o grupo de Inteligência Artificial de Stanford iniciou o desenvolvimento de uma linguagem de programação explícita baseada na linguagem ALGOL (Algorithmic Language -Linguagem Algorítmica-) e incorporava construções avançadas para o controle de vários braços simultaneamente. Assim nasceu o AL, projetado para facilitar a programação de tarefas de montagem, aproveitando a grande experiência que esse laboratório possuia na área de sensoreamento.

Por sua vez a IBM, desenvolvia naqueles dias no Centro de Pesquisa "Thomas J. Watson" em New York importantes experiências que apontavam para a criação de diversas linguagens para o controle de manipuladores do tipo cartesiano. As primeiras linguagens foram EMILY e ML, as quais foram usadas com bastante sucesso em diferentes tarefas de montagem.

Dois outros laboratórios usaram as linguagens de tipo comercial como plataforma para a criação de sub-rotinas apontando ao controle de robôs. O SRI desenvolveu o RPL, uma linguagem de programação voltada à pesquisa e que foi distribuida como um serviço para todos os afiliados ao Programa Industrial SRI (Stanford Research Institute). RPL (Robot Programming Language) enfatiza o uso de subrotinas de propósito específico que são chamadas por um programa de propósito geral, sendo projetada para facilitar o controle de variadas máquinas que formam parte de uma célula robotizada (robôs, máquinas ferramentas, sensores, etc.).

O Jet Propulsion Laboratory desenvolveu o JARS, um sistema de programação baseado no PASCAL, para o controle de robôs usados na montagem de estruturas para células solares.

A primeira linguagem de programação comercialmente disponível foi o VAL, criado em 1979 pela UNIMATION. VAL (Versatile Assembly Language) foi desenvolvida como uma extensão da linguagem de programação BASIC por um graduado de Stanford que tinha tido um contato muito direto com a AL. Inicialmente VAL era oferecida com o manipulador PUMA ("Programmable Universal Machine for Assembly"), porém atualmente VAL está disponível para

muitos manipuladores da UNIMATION. Mais recentemente foi liberada ao mercado uma nova versão chamada VAL II, a qual é uma extensão das capacidades de sua antecessora.

Embora não seja considerada como uma verdadeira linguagem do tipo textual, os robôs do tipo Cincinatti Milacron T3 fornecem ao usuário elementos semelhantes a uma linguagem. Desde a sua introdução, no final da década de 70, a série T3 tem sido muito utilizada, particularmente na indústria aeroespacial.

Recentemente outra opção foi desenvolvida para o T3. Esta permite ao usuário obter uma cópia em disco das sequências programadas usando o sistema "teach-in" ou o painel de controle.

A característica mais importante deste tipo de linguagens é a extensibilidade. Isto significa que a linguagem pode ser extendida ou aprimorada pelo usuário para dar conta dos requisitos de futuras aplicações, futuros dispositivos e/ou futuros robôs, todos podendo ser mais sofisticados do que na época em que a linguagem foi inicialmente projetada. Significa também que a linguagem pode ser expandida desenvolvendo comandos, sub-rotinas e macros, que não foram incluídos no conjunto inicial de instruções **[MEND-87]**.

As linguagens deste tipo retêm todas as funções essenciais da linguagem usada como plataforma de desenvolvimento. Além das estruturas clássicas incorporam novos tipos de dados específicos para realizar a programação de aplicações robóticas. Estas estruturas podem ser classificadas em **[POLL-87]** :

- Dados geométricos.
- Instruções de movimento, com parâmetros de interpolação, velocidade, aceleração, precisão, etc.
- Instruções paralelas, para vários robôs cooperativos.
- Processador de sinais provenientes ddesde sensores.
- Instruções para controlar o fluxo de sinais dos dispositivos I/O.

Posteriormente foram desenvolvidas as linguagens de programação específicas; por específicas podemos entender as que possuem uma sintaxe que não é herdada de nenhuma

linguagem anterior, que possuem compiladores ou interpretadores especialmente projetados para essa linguagem, e que portanto tem uma estrutura dedicada à definição de tarefas ou operações robotizadas.

Em 1981 foram introduzidos duas linguagens de programação para robôs industriais desse tipo, a AUTOMATIX desenvolveu o RAIL (Robotic Automatix Incorporated Language) para controlar as operações de inspeção, soldagem e montagem. Com o apoio de U.S. Air Force ICAM, a McDonell-Douglas desenvolveu o MCL (Manufacturing Control Language), extensão da linguagem APT (Automatically Programmed Tool -Programação para Comando Numérico de Máquinas-Ferramenta), para a programação "off-line" de robôs usando uma base de dados CAD. Esta linguagem está sendo oferecida comercialmente nos Estados Unidos.

Em 1982 outras duas novas linguagens foram anunciadas no mercado. AML criada pela IBM Corporation e HELP (High Level Procedural Language) pela General Electric Company. AML (A Manufacturing Language), de acordo com os seus criadores é uma poderosa linguagem baseada num conjunto de simples instruções para o uso de programadores experientes na programação dos manipuladores da série IBM 75xx.

Por sua vez HELP foi criada para realizar a programação dos manipuladores da série A12 Allegro Robot System, do tipo cartesiano, voltados para operações de montagem. A principal vantagem dessa linguagem é que ela pode ser configurada para um máximo de quatro braços, tendo no máximo doze graus de liberdade.

Este tipo de linguagens de programação traz algumas desvantagens para os usuários de diversos tipos de robôs :

○ Cada sistema robotizado ou cada robô, precisa de um programador especificamente treinado para tal linguagem.

○ O processo de transferência de um programa existente de um robô para outro, é equivalente à implementação pela segunda vez das tarefas, mesmo tendo estes, características cinemáticas similares.

○ Não existe possibilidade de transferência do conhecimento de coordenadas de um sistema de

controle de um robô para outro.

A solução para tais problemas pode ser encontrada na padronização das interfaces entre sistemas de programação e os controladores de diversos robôs. Isto se traduzirá nas seguintes vantagens :

○ Os programas escritos em um sistema especial de programação poderão ser usados para controlar manipuladores de diversos tipos.

○ Um especialista em programação para um robô específico se converterá em um especialista amplo na programação de robôs.

○ Um robô poderá usar programas provenientes de qualquer sistema de programação.

Com maior frequência são desenvolvidas novas linguagens de programação para robôs industriais, Rembold relacionou em 1987 mais de 100 linguagens de programação para robôs **[REMB-87]**, demonstrando um mercado amadurecido e amplo para as diversas necessidades dos usuários.

2.2.2. Programação Implícita

O método de programação implícita é semelhante à programação explícita, o usuário programa uma tarefa através de comandos de altíssimo nível, ou dito de outra forma, define de maneira abstrata as tarefas a serem realizadas pelo manipulador.

Posteriormente esta representação abstrata é transferida a uma representação concreta ou de mais baixo nível usualmente com a ajuda de ferramentas de Inteligência Artificial e/ou sistemas especialistas, gerando a sequência de ações ou instruções a serem entendidas e executadas

pelo manipulador.

A filosofia da programação implícita caracteriza-se pela descrição geométrica das tarefas, através do uso de modelos ou referenciais fixados em cada um dos objetos existentes na célula que está sendo usada na aplicação. A este procedimento chama-se "modelamento do mundo" e usualmente, é feito com a ajuda de funções gráficas interativas, que na maioria das vezes fazem parte de um pacote CAD.

Este tipo de sistemas podem possuir um ambiente completo de programação, composto por:

- Uma base de dados CAD
- Sistema de modelamento geométrico
- Sistema de planejamento de tarefas
- Um sistema de sensoreamento de alto nível, inclusive com a capacidade de visão computacional.

Tais ambientes integrados são essenciais para fornecer um certo grau de inteligência que reduza ao mínimo a intervenção do ser humano na definição das tarefas do manipulador.

As linguagens de programação implícita podem ser classificadas em dois níveis dependendo da forma como é feita o "modelamento do mundo" e das tarefas a serem realizadas pelo manipulador.

Estas duas categorias são as linguagens a nível de objeto, no qual as ações são descritas em termos de relações entre os objetos descritos. Rondeau e ElMaraghy **[ROND-90]** descrevem este nível de programação e citam como exemplos desta filosofia de programação implícita os seguintes casos : LM-GEO, RAPT (Robotic APT), AUTOPASS (Automatic Parts Assembly System) e WROPE.

A segunda categoria é a que descreve as ações através de sub-tarefas diretamente. Esta descrição consiste basicamente de três partes :

- Descrição do estado inicial do "mundo".
- Uma sequência de relações que descrevem a tarefa ou estado final.
- Uma descrição das possíveis ações do robô.

O objetivo é determinar automaticamente a sequência de operações de alto nível, com as quais, pode ser "atingido" o "estado final do mundo", e transferir essas operações a um programa robótico compatível com determinado controlador. Constiuem alguns exemplos constituem AUTOFIX **[WECK-88]** e ROBOPLAN **[ELMA-91]**.

É evidente que no futuro este método irá a ocupar a maior parte do mercado dos robôs industriais, porém ainda é necessário uma considerável quantidade de esforço para desenvolver este tipo de sistemas em escala industrial.

2.2.3 Método Híbrido

Alguns autores consideram a existência de um método híbrido **[STOR-87]**, que mistura a programação "off-line" e a aquisição de dados espaciais via "teach-in", permitindo diminuir, mas não evitar o uso do robô como ferramenta de programação. É assim que, embora sejam usados linguagens e sistemas de programação muito poderosos e flexíveis, os programas devem ser finalmente testados e depurados usando o robô. Adicionalmente o robô é usado na tarefa de aquisição das coordenadas, ajudando na transferência de informações do modelo real ao programa textual. A seguir são apresentados vários quadros comparativos das características das principais linguagens de programação para robôs **[GRUV-84]**:

Tabela 2.1. Modalidade de Linguagem.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Textual	X	X	X	X	X	X	X	X
Via Menú		X						

Tabela 2.II. Tipo de Linguagem.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Subrotinas				X			X	
Extensão					X			
Nova	X	X	X			X		X

Tabela 2.III. Tipos de Dados Geométricos.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
"Frame"	X			X	X	X		X
Espaço de Junta		X		X				
Vetor	X	X		X	X			
Rotação	X			X	X			
Trajetória						X		

Tabela 2.IV. Representação e Especificação das rotações.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Matriz Rotações	(1)			(4)				
Âng.c/r a vetor	X			(4)				
Âng.de Euler	(2)	X		X		X		X
Roll-Pitch-Yaw	(3)		X					

1) AL apresenta as rotações na forma de matriz.
2) AL aceita diretamente a especificação de uma orientação através dos três ângulos de Euler ou através de um ângulo em relação a um vetor.
3) As orientações em AL podem também ser especificadas através dos ângulos Roll-Pitch-Yaw
4) Normalmente JARS não apresenta rotações nessa forma, porém o usuário pode escrever uma rotina para imprimí-las, já que JARS usa internamente o formato de matrizes de rotação.

Tabela 2.V. Capacidade de Controlar Múltiplos Braços.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Um braço		X		X		X	X	X
Vários braços	X	(1)	X		X			

1 Ainda não disponível comercialmente.

Tabela 2.VI. Estruturas de Controle

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Labels		X	X	X	X		X	X
If - Then	X	X	X	X	X	X	X	X
If-Then-Else	X	X	X	X	X	X	X	
While-Do	X	X	X	X	X	X	X	
Do-Until	X	X		X		X	X	
Case	X			X		X	X	
For	X	X		X		X	X	
Begin-End	X	X		(1)	(2)			
Cobegin-Coend	X		(3)		(2)			
Proc/Funct/Subrot	X	X	X	X	X	X	X	

1 Já que JARS é uma linguagem baseada em estruturas adicionadas ao PASCAL, JARS possue todas as estruturas de controle dele.
2 MCL pode chamar estruturas em paralelo.
3 HELP permite a ativação simultânea de várias tarefas.

Tabela 2.VII. Modos de Controle.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Posição	X	X	X		X	X	X	X
Força	X							(1)
Por complacência	X			(2)				

1 Em VAL-II
2 Atualmente implementando-se no Jet Propulsion Laboratory

Observação : Estes modos de controle podem ou não existir no sistema que está sendo controlado.

Tabela 2.VIII. Capacidades Sensoriais.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Visão	X	(1)	X	X	X	X	X	X
Força	X	X	X					(2)
Proximidade	X	X		X	X	X	X	X

1 Ainda não disponível comercialmente
2 Apenas para o VAL-II

Tabela 2.IX.Módulos de Suporte.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Editor	(1)	X	(3)	(3)		X	X	X
Ger.Arquivos	(1)	X	(3)	(3)		X	X	X
Interpretador	X	X	X			X	(1)	X
Compilador	X			X	X		X	
Simulador	X	(2)			X			
Macros	X		X		X			
Bibliotecas	X	X						
Arq.de Comando				X			X	
Ajuda	X							
Tutoriais		X						

1 Baseado no sistema operacional PDP-10
2 Desenvolvido pela IBM no Watson Research Center
3 JARS e HELP usam o suporte do sistema operacional RT-11

Tabela 2.X. Ferramentas de Depuração.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Passo a passo		X	X			X	X	
Breakpoints	X	X				X	X	
Trace		X	X		X		X	
Dump	X		X		X		X	

Tabela 2.XI. Linhas de Sinais.

	AL	AML	HELP	JARS	MCL	RAIL	RPL	VAL
Entradas Binárias	0	64	*	0	242	6	32	32
Saídas Binárias	0	64	*	2	242	10	32	32
Entradas Analógicas	64	0	*	0	242	0	32	0
Saídas Analógicas	4	0	0	0	242	0	64	0

* Sob encomenda

3. O MANIPULADOR

O IBM 7535 é um robô industrial do tipo SCARA (Selective Compliance Assembly Robot Arm ou Braco de Robô de Montagem por Complacência Seletiva) de 3 1/2 eixos controlados, muito usado em uma variedade de operações de montagem leve. O tipo de operações mais comuns são as do tipo "pega e põe" ("pick and place") e operações de paletização **[CRAI-85]** (Figura 3.1.)

As duas primeiras juntas, do ombro e cotovelo, são do tipo rotacional e giram em torno de eixos verticais, movimentando o braço num plano horizontal (XY), propiciando substancial rigidez para o robô no sentido vertical, mas complacência no plano horizontal **[IBM-83]**.

O intervalo angular das duas primeiras juntas é o seguinte :

junta 1	$0 < \theta_1 < 200^\circ$
junta 2	$0 < \theta_2 < 160^\circ$

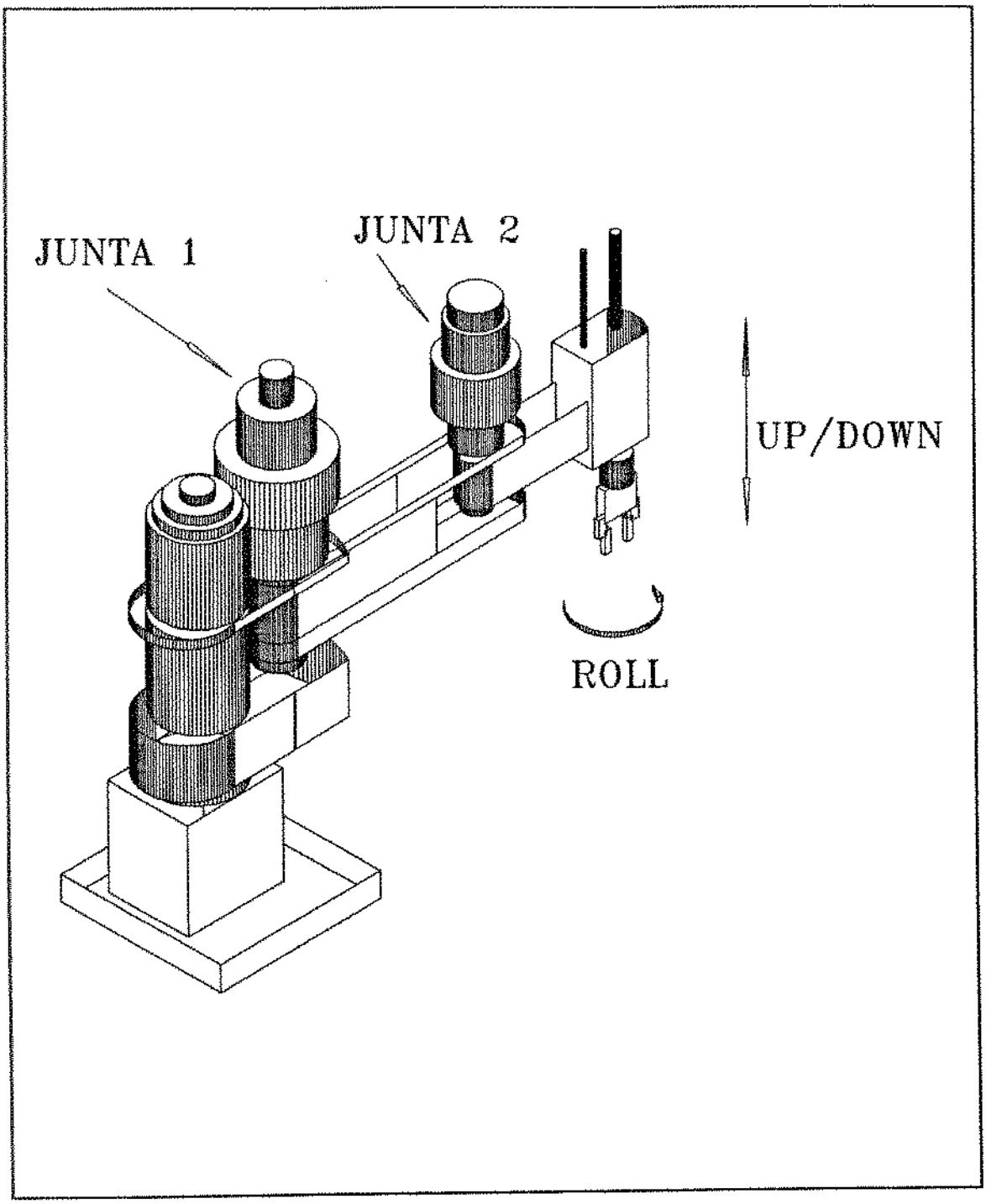


Fig.3.1. Esquema geral do robô SCARA (IBM 7535)

A figura 3.2. ilustra ambos os intervalos angulares.

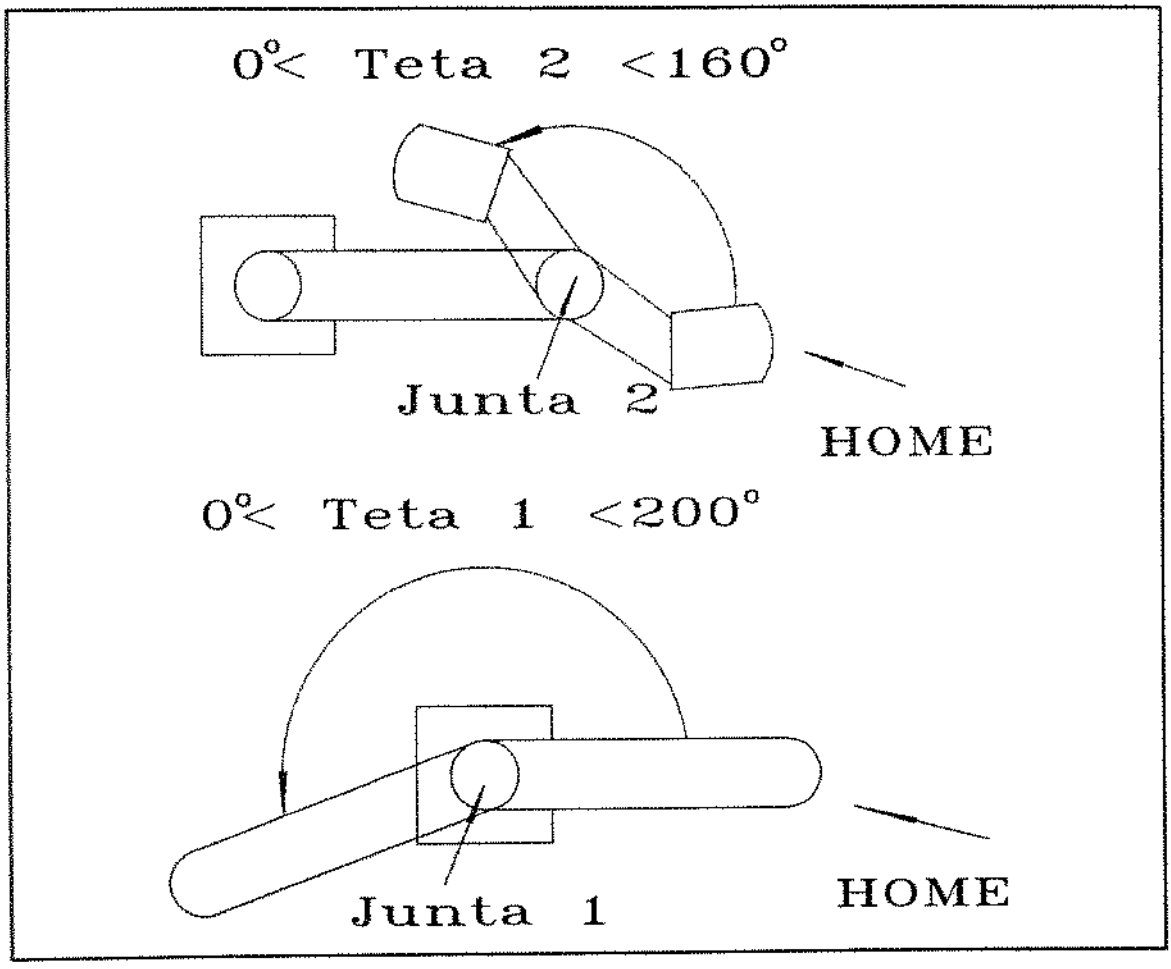


Fig. 3.2. Intervalos angulares das duas juntas.

Segundo estes intervalos permitidos para cada uma das juntas determina-se o "envelope de trabalho" para o manipulador. Uma projeção no plano horizontal XY é mostrada na figura 3.3.

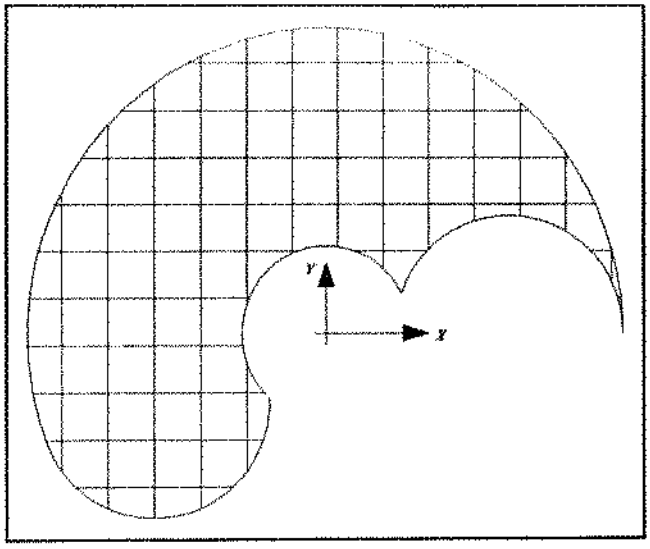


Fig. 3.3. Espaço de trabalho do SCARA

A denominação de 3 1/2 eixos controlados prende-se ao fato de que o terceiro elo tem apenas duas posições permitidas (UP e DOWN) acessadas por software, não sendo possível o posicionamento contínuo do atuador no eixo vertical, este posicionamento será então apenas discreto.

O último grau de liberdade está dado pelo movimento rotacional da garra, este movimento rotacional é chamado de ângulo "roll" é e feito em função do eixo vertical Z (Figura 3.4)

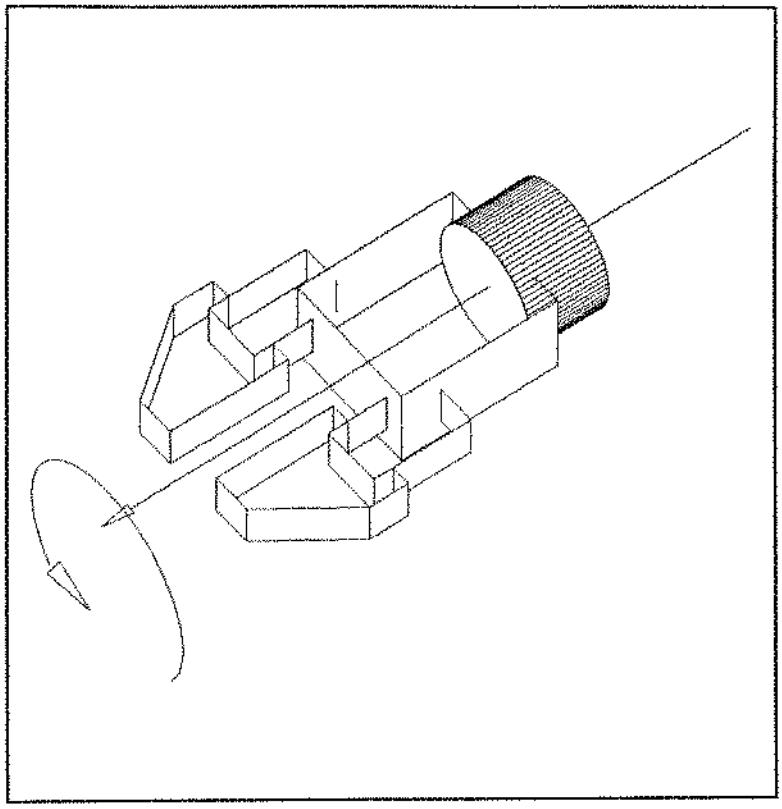


Fig. 3.4. Ângulo roll da garra.

3.1. Modelamento do robô

O fundamento de toda simulação é um modelo matemático do processo real. Através deste modelo são descritas as entidades que fazem parte do processo que se deseja simular, as relações entre eles e as restrições que atuam sobre cada um dos elementos e sobre o conjunto. Kopacek e Troch **[KOPA-88]** fizeram um estudo da estrutura necessária para modelar o robô e a interação com o seu ambiente, assim os tipos de modelos requeridos para representar um robô industrial e as suas operações geralmente são divididos segundo as seguintes classes **[MEIJ-88]**:

- Modelo geométrico
- Modelo cinemático
- Modelo tecnológico

O modelo geométrico contém toda a informação sobre a localização espacial do manipulador e de todos os corpos existentes no sistema durante a sequência de movimentos programados. Para este propósito, as geometrias das entidades, que frequentemente são de uma alta complexidade, são aproximadas a modelos simplificados **[WOZN-88]**.

Durante a simulação estes datos são continuamente adaptados às alterações das posições do manipulador e dos outros objetos, constituindo-se na base de dados geométrica para as representações gráficas subsequentes.

O tipo e estrutura do modelo geométrico tem um grande impacto na precisão do processo de simulação, em particular nas rotinas de deteção automática de colisões. Por outro lado, a quantidade das informações geométricas influirão ditretamente no desempenho computacional e, é claro, no nível de detalhamento da representação gráfica **[MILB-89]**.

O modelo cinemático armazena as informações necessárias para determinar a posição e orientação do efetuador ao executar um determinado movimento. Esta posição e orientação depende consequentemente do tipo dos eixos individuais, que são descritos através de modelos rotacionais e/ou traslacionais e das relações entre cada um desses eixos e os eixos seguintes na cadeia cinemática do manipulador **[CRAI-85]**.

Finalmente a informação tecnológica representa todos os dados relativos ao processo. Assim, neste modelo estarão especificados os valores máximos de movimentação para cada junta, sendo possível detectar posicionamentos não permitidos durante a simulação; a especificação das velocidades e acelerações para cada junta; a informação sobre as características dinâmicas do manipulador e das juntas, etc.

A seguir são apresentados os modelos construídos para realizar a representação do robô usado neste trabalho :

3.1.1. Modelamento Cinemático

A fim de criar um modelo para representar o movimento, é necessário usar técnicas para calcular e conhecer a posição do braco em pontos no tempo a partir de certa informação conhecida.

Dadas as características de projeto do manipulador que usamos neste trabalho é necessário apenas considerarmos as duas primeiras juntas, pois são as que determinam a posição da garra no plano horizontal **[HEUB-90]**.

Assim, a posição do extremo do braço pode ser representada por dois métodos, o primeiro utiliza os ângulos das duas juntas Θ1 e Θ2. Isto é conhecido como a representação no espaco de junta, segundo ela podemos definir um ponto Pj, como :

$$Pj = (\Theta_1, \Theta_2)$$

O outro modo de definir a posição do braço é no espaco cartesiano. Isto envolve o uso de um sistema de coordenadas cartesianas que é externo ao robô. A origem do sistema de eixos cartesianos está localizada na base do robô. A posição do fim do braço, Pw, seria definida no espaço bidimensional como :

$$Pw = (X, Y)$$

Este tipo de representação "neutra" é a mais apropriada para nossos objetivos, já que todos os comandos que envolvem movimentos e posições na linguagem AML estão definidos usando este tipo de representação.

Porém, a representação gráfica do manipulador é mais simples de ser feita através da representação no "espaço de junta", pois isto envolve apenas, o cálculo de rotações em torno do eixo vertical do sistema de referência, fixado na base do manipulador (ambas juntas são rotacionais).

Para usarmos ambas as representações, devemos transformar de uma para a outra representação. Ir desde o "espaço de junta" para o "espaço cartesiano" é chamado transformação direta, e ir do "espaço cartesiano" para o "espaço de junta", transformação inversa.

3.1.1.1. Transformação Direta

Podemos determinar a posição da extremidade do braço no espaço cartesiano definindo um vetor para o elo 1 e outro para o elo 2 (Figura 3.5) :

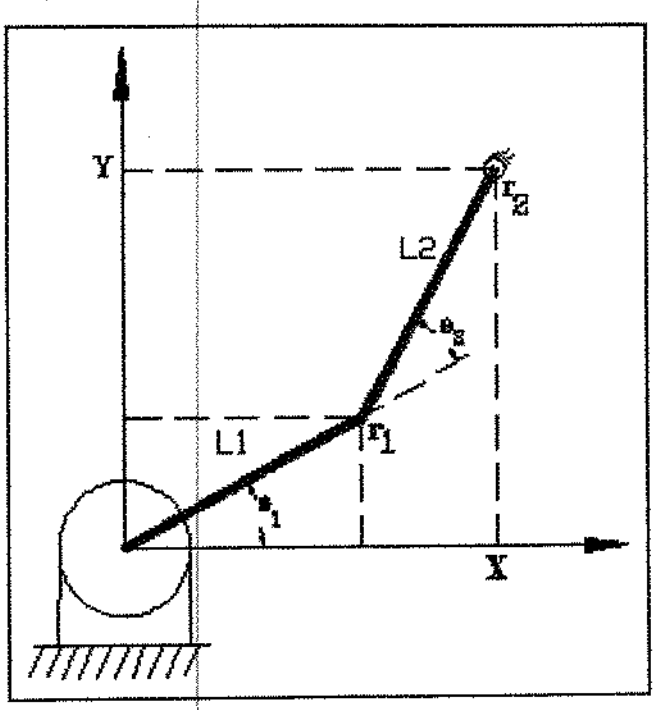


Fig. 3.5. Representação geométrica do Modelo Direto.

$$r_1 = [\, L_1 \cos \Theta_1 \, , \, L_1 \operatorname{sen} \Theta_1 \,] \qquad (3.1)$$

$$r_2 = [\, L_2 \cos (\Theta_1 + \Theta_2) \, , \, L_2 \operatorname{sen} (\Theta_1 + \Theta_2) \,] \qquad (3.2)$$

A adição vetorial das equações (3.1) e (3.2) gera as coordenadas X e Y da extremidade do braço (ponto Pw) no espaço cartesiano :

$$X = L_1 \cos \Theta_1 + L_2 \cos (\Theta_1 + \Theta_2) \qquad (3.3)$$

$$Y = L_1 \operatorname{sen} \Theta_1 + L_2 \operatorname{sen} (\Theta_1 + \Theta_2) \qquad (3.4)$$

Assim conhecendo os ângulos das duas primeiras juntas, poderemos determinar a posição no plano XY (espaço cartesiano) do atuador do robô.

3.1.1.2. Transformação Inversa

Como já foi dito anteriormente, desde o ponto de vista da linguagem, é mais importante poder derivar os ângulos das juntas dada a posição do orgão terminal no espaço cartesiano.

No caso das duas primeiras juntas, ou seja considerando os dois primeiros elos, deslocando-se no plano horizontal, existem duas configurações possíveis para alcançar o ponto (X,Y), como mostrado na figura 3.6.

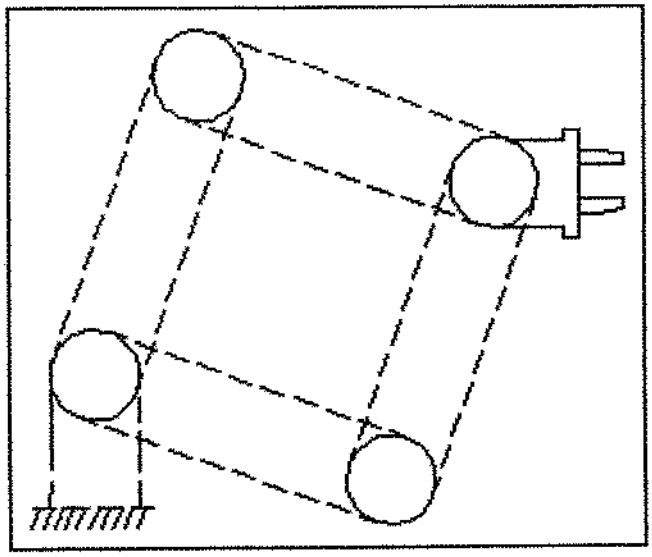


Fig.3.6. Configurações alternativas para um mesmo ponto.

Considerando as restrições de projeto do IBM 7535 esta situação não existirá já que as juntas só podem rotacionar com ângulos positivos [IBM-83]. Assim, existirá apenas uma única solução para cada posição no espaço de trabalho, eliminando a redundância do seguinte modelo:

Usando as seguintes identidades trigonométricas

$$\cos (A + B) = \cos A \cos B - \text{sen } A \text{ sen } B$$
$$\text{sen} (A + B) = \cos A \text{ sen } B + \text{sen } A \cos B$$

podemos reescrever as equações (3.3) e (3.4) como

$$X = L_1 \cos\theta_1 + L_2 (\cos\theta_1\cos\theta_2) - L_2 (\text{sen}\theta_1\text{sen}\theta_2) \quad (3.7)$$
$$Y = L_1 \text{sen}\theta_1 + L_2 (\text{sen}\theta_1\cos\theta_2) + L_2 (\cos\theta_1\text{sen}\theta_2) \quad (3.8)$$

Elevando ao quadrado ambos os termos e somando as duas equações, teremos :

$$\cos\theta_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2 L_1 L_2} \quad (3.9)$$

Definindo α e ß como na figura 3.7. obtemos

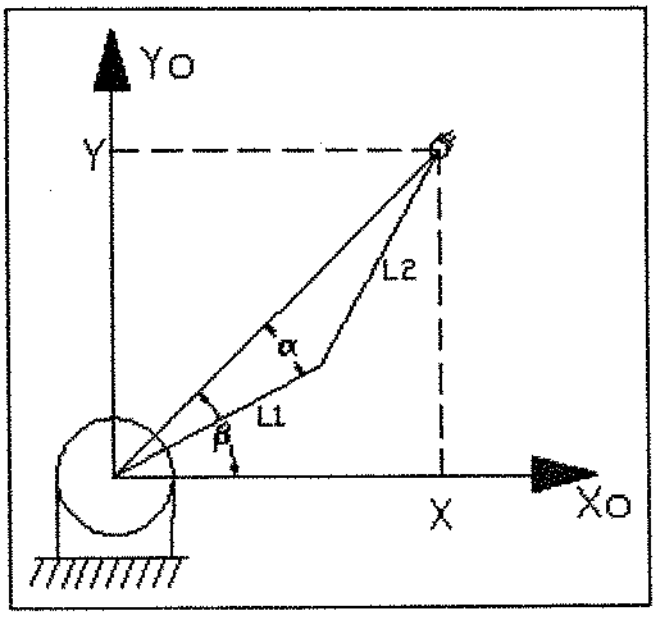


Fig.3.7. Ângulos auxiliares.

$$\tan \alpha = \frac{L_2 \operatorname{sen} \Theta_2}{L_2 \cos \Theta_2 + L_1} \qquad (3.10.)$$

$$\tan \beta = Y / X \qquad (3.11.)$$

Usando a identidade trigonométrica :

$$\tan (A\text{-}B) = \frac{\tan A - \tan B}{1 + \tan A \tan B}$$

Obtemos

$$\tan \Theta_1 = \frac{[\, y\,(L_1 + L_2 \cos \Theta_2) - x\,(L_2 \operatorname{sen} \Theta_2)\,]}{[\, x\,(L_1 + L_2 \cos \Theta_2) + y\,(L2 \operatorname{sen} \Theta_2)\,]} \qquad (3.12)$$

Assim poderemos conhecer os ângulos requeridos das juntas para colocar o braco na posição (X,Y) no espaço cartesiano através das seguintes equações :

$$\Theta_1 = \tan^{-1} \frac{[\, y(L_1 + L_2 \cos \Theta_2) - x(L_2 \operatorname{sen} \Theta_2)\,]}{[\, x(L_1 + L_2 \cos \Theta_2) - y(L_2 \operatorname{sen} \Theta_2)\,]} \qquad (3.13)$$

$$\theta_2 = \cos^{-1} \frac{x_2 + y_2 - L_1^2 - L_2^2}{2L_1L_2} \qquad (3.14)$$

3.2. Modelamento Geométrico para Representação Gráfica

Este capítulo apresenta os métodos e procedimentos usados na representação gráfica do manipulador. Estes procedimentos baseiam-se numa adaptação dos parâmetros de Denavit-Hartenberg, e na utilização de prismas regulares para a representação esquemática da cadeia cinemática do robô.

Cada um dos três elos e a garra do manipulador estão representados por um prisma regular com os seus vértices definidos em relação a um sistema de referência local, próprio de cada articulação, conforme mostra a figura 3.8.[ERTH-91].

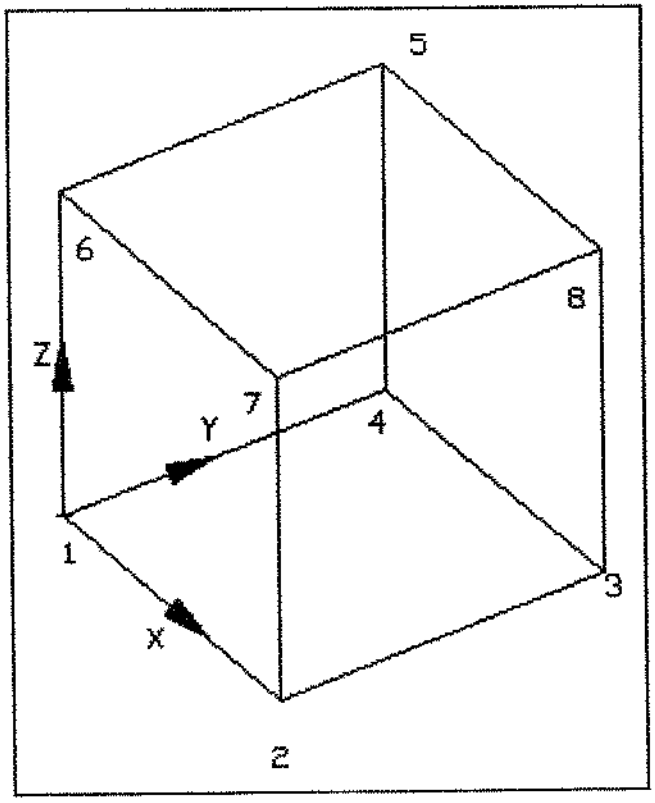


Fig.3.8. Distribuição dos vértices para cada elo.

A representação interna dos elos obedece a duas tabelas principais, a tabela de vértices e a tabela de arestas. Na tabela de vértices são armazenados cada um dos oito vértices necessários na definição de um paralelepípedo. Estes vértices têm a forma de vetores em três dimensões e são identificados da seguinte forma :

$$P_{i,j}\ (x,y,z)$$

Onde i indica cada um dos vértices. i = 1 , ... , 8 e j identifica o número do elo ao qual os vértices pertencem.

Assim, por exemplo, o primeiro elo (No 1), com as seguintes dimensões :

altura : 45 mm
largura : 27 mm
comprimento : 67 mm

Terá a seguinte representação interna :

Tabela 3.I. Representação interna de um elo do manipulador.

VÉRTICE	X	Y	Z
1	0.0	0.0	0.0
2	27.0	0.0	0.0
3	27.0	67.0	0.0
4	0.0	67.0	0.0
5	0.0	67.0	45.0
6	0.0	0.0	45.0
7	27.0	0.0	45.0
8	27.0	67.0	45.0

Adicionada a esta tabela existem outras duas tabelas com a informação referente a como os vértices estão unidos para formar cada uma das doze arestas que fazem parte de um elo, no

primeiro caso e para conformar a garra no segundo caso.

A representação interna destes dados, no caso dos elos, tem a seguinte forma :

Tabela 3.II. Representação interna das arestas de um elo.

Número Aresta	Vértice Inicial	Vértice Final
1	1	2
2	2	3
3	3	4
4	4	1
5	1	6
6	6	7
7	7	8
8	8	5
9	5	6
10	5	4
11	8	3
12	7	2

A seguinte tabela mostra a relação entre os vértices usada para representar graficamente a garra.

Tabela 3.III Representação interna das arestas da garra.

Número Aresta	Vértice Inicial	Vértice Final
1	1	6
2	6	7
3	7	2
4	2	1
5	6	5
6	5	8
7	8	3
8	3	4
9	4	5
10	8	7

Para representar a cadeia cinemática do robô, são fixados sistemas de coordenadas em cada um dos elos e no atuador. Estes sistemas são definidos e fixados de maneira que coincidam com os referenciais locais usados na definição dos vértices de cada um dos elos. Eles são numerados começando pela base imóvel do braço, ao qual é fixado o referencial 0. O primeiro elo móvel recebe o referencial 1, e assim em diante até o segmento livre - o atuador - que tem fixado a ele o sistema referencial 4. A localização destes referenciais é mostrado na figura 3.9.

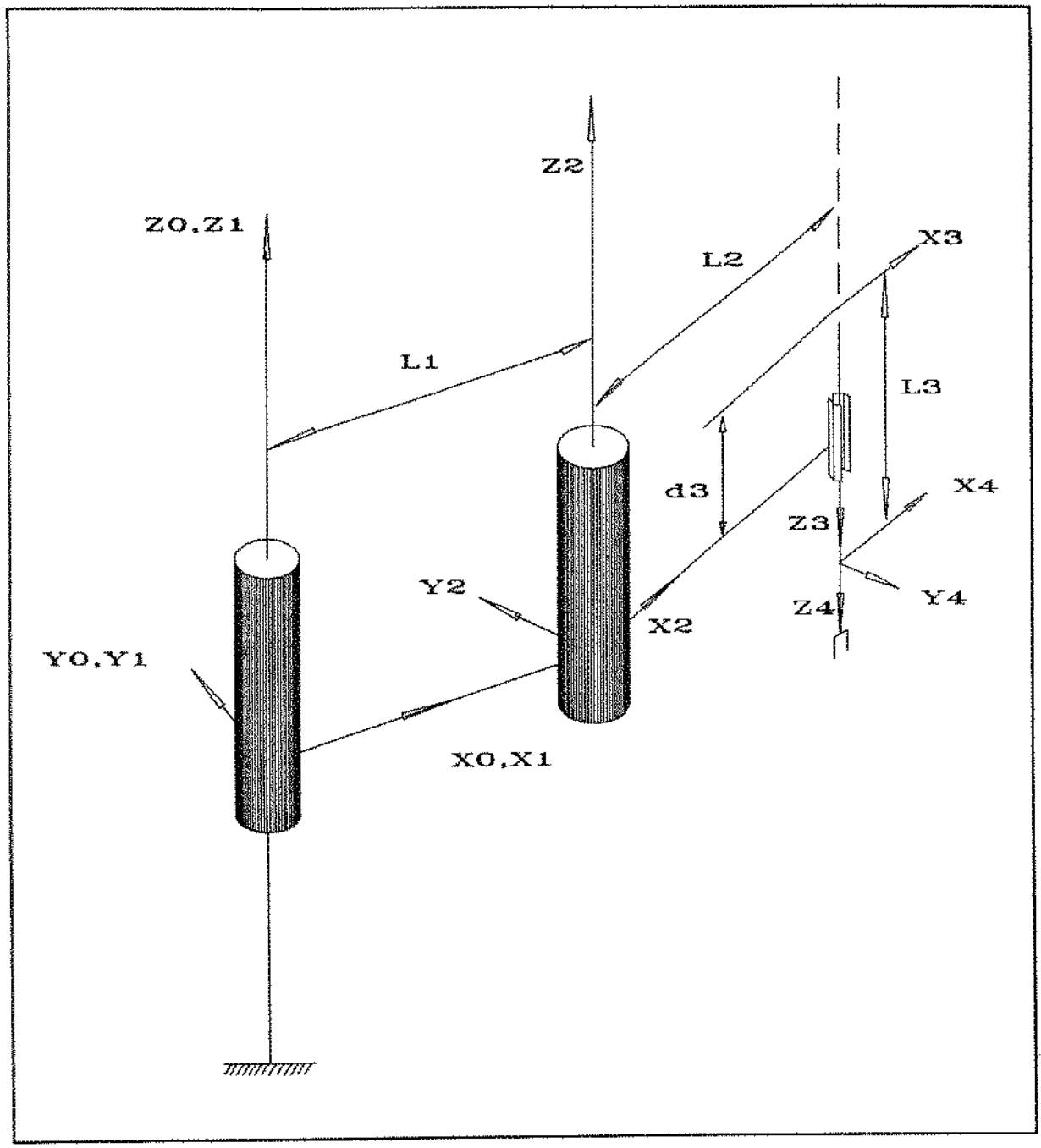


Fig.3.9. Esquema de localização dos referenciais no manipulador.

Considerando a posição dos referenciais especificada na figura são obtidos os parâmetros de Denavit-Hartemberg para o manipulador SCARA [DENA-55]. Estes parâmetros são detalhados na seguinte tabela :

Tabela 3.IV .Parâmetros de Denavit-Hartemberg para um robô SCARA.

JUNTA	Θ	α	a	d
1	Θ_1	0º	0	0
2	Θ_2	0º	L_1	0
3	0	180º	L_2	$-d_3$
4	Θ_4	0º	0	L_3

A configuração do braço para cada posição a ser representada, está determinada por uma tabela gerada pelo processo de interpretação do programa AML. Esta tabela contém as posições angulares relativas do elo 1, respeito do sistema referencial 0 (fixado na base), do elo 2, com respeito ao sistema referencial fixado no elo 1 e o ângulo Roll da garra, medido com respeito ao referencial fixado no terceiro elo do manipulador, e a posição vertical (medida no eixo Z) do terceiro elo com respeito ao elo 2.

Para realizar a representação gráfica do manipulador deve-se converter cada uma das coordenadas dos elos desde as definidas em função do referencial local a uma representação relativa ao referencial absoluto (referencial 0). Usando os parâmetros de Denavit-Hartemberg pode-se determinar as transformações que definirão, de uma maneira geral, as posições relativas ao sistema de referencia {i}, com respeito de um outro referencial {i-1}.

Na representação de cada um dos elos com respecto de outro, em particular o anterior na cadeia cinemática, realiza-se uma transformação de cada um dos pontos que compõem cada elo através de uma matriz de transformação homogênea. Esta matriz é obtida em função dos quatro parâmetros D-H para cada elo.

Cada uma destas transformações representa de maneira geral duas traslações e duas rotações, como expresso na seguinte equação :

$$^{i-1}_{i}T = R_x(\alpha_{i-1})\, D_x(a_{i-1})\, R_z(\theta_i)\, D_z(d_i) \qquad (3.15)$$

ou

$$^{i-1}_{i}T = \text{Screw}_x(a_{i-1},\alpha_{i-1})\, \text{Screw}_z(d_i,\theta_i) \qquad (3.16)$$

Onde a notação ScrewQ(r,γ) define uma traslação ao longo do eixo Q de distância r, e uma rotação em torno do mesmo eixo de um ângulo γ.

Fazendo as multiplicações em (3.15) ou em (3.16) obtém-se a forma geral da transformação $^{i-1}_{i}T$:

$$^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -\text{sen}\theta_i & 0 & a_{i-1} \\ \text{sen}\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\text{sen}\alpha_{i-1} & -\text{sen}\alpha_{i-1}d_i \\ \text{sen}\theta_i\text{sen}\alpha_{i-1} & \cos\theta_i\text{sen}\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.17)$$

Assim, usando os parâmetros da tabela 3.IV e a equação (3.17) pode-se determinar as quatro transformações necessárias para definir a representação dos pontos pertencentes a um elo em relação ao elo anterior na cadeia cinemática. Estas quatro transformações são descritas a seguir:

$$^{0}_{1}T = \begin{bmatrix} \cos\theta_1 & -\text{sen}\theta_1 & 0 & 0 \\ \text{sen}\theta_1 & \cos\theta_1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$
{}^{1}_{2}T = \begin{matrix} \cos\Theta_2 & -\text{sen}\Theta_2 & 0 & L_1 \\ \text{sen}\Theta_2 & \cos\Theta_2 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}
$$

$$
{}^{2}_{3}T = \begin{matrix} 1 & 0 & 0 & L_2 \\ 0 & -1 & 0 & 0 \\ 0 & -1 & -1 & d_3 \\ 0 & 0 & 0 & 1 \end{matrix}
$$

$$
{}^{3}_{4}T = \begin{matrix} \cos\Theta_4 & -\text{sen}\Theta_4 & 0 & 0 \\ \text{sen}\Theta_4 & \cos\Theta_4 & 1 & 0 \\ 0 & 0 & 0 & L_3 \\ 0 & 0 & 0 & 1 \end{matrix}
$$

Obtendo cada uma das quatro matrizes de transformação homogêneas (uma para cada junta) a partir da informação existente na tabela gerada pelo pré-processador, pode-se calcular, a posição de cada vértice respeito do referencial absoluto (fixo na base do manipulador).

A seguinte equação permite realizar estas transformações para todos os pontos que definem a totalidade do manipulador :

$$
P_{i,0} = {}^{0}_{1}T * {}^{1}_{2}T * \ldots * {}^{j-1}_{j}T * P_{i,j}
$$

Onde i representa cada um dos vértices de cada elo (i = 1, ... ,8) e j representa cada elo (j = 1, ... , 4). Note-se que estas multiplicações são de uma ou várias matrizes por um vetor, portanto deve-se considerar a ordem da multiplicação, já que o produto de matrizes ou de uma matriz por um vetor não é comutativo.

O processo de representação gráfica lê cada um dos ângulos existentes na tabela e avalia

a equação gerando a nova posição de todos os vértices dos elos e da garra com as suas coordenadas definidas com respecto da base do robô. Uma vez que esse processo é terminado, as coordenadas são unidas através de linhas seguindo a informação da tabela de arestas.

Finalmente estas linhas são exibidas na tela, completando a representação do manipulador na configuração dada.

4. ESTRUTURA DO SISTEMA

O sistema é orientado ao usuário e utiliza a entrada de dados em forma conversacional e interativa, através de menús que levam ao usuário a quatro módulos principais :

- Gerenciador de "Lay-outs".
- Pré-processador de programas em AML/E.
- Simulador Gráfico.
- Deteção Automática de Colisões.

A figura 4.1.mostra o Menú Principal do sistema :

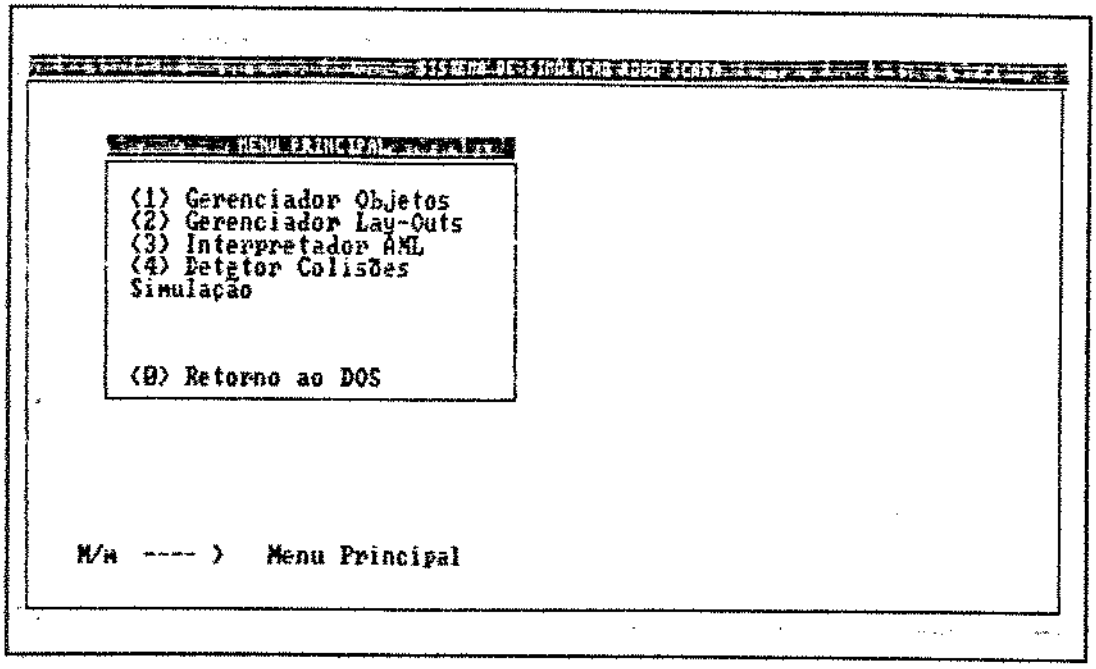


Fig. 4.1. Menú principal do sistema.

A estrutura lógica do sistema é mostrada na figura 4.2.. Nela pode-se observar o relacionamento entre o usuário e cada um dos módulos principais do sistema, e o fluxo de informações desde e para o ao interior dele.

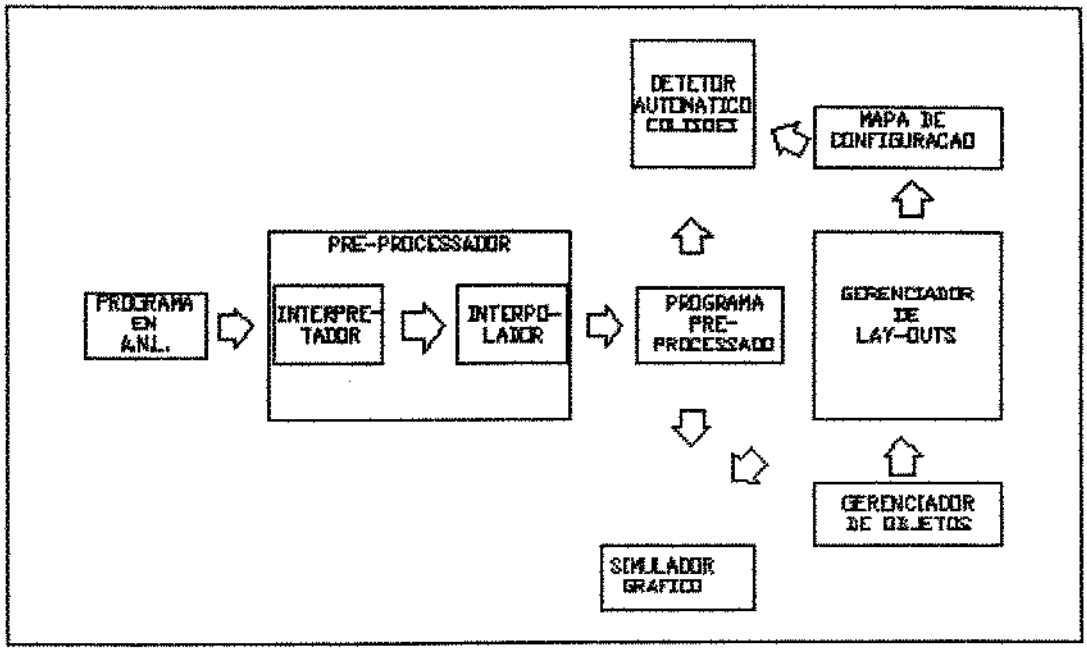


Fig. 4.2. Estrutura geral do sistema.

Uma vez editado o programa em AML o usuário estará capacitado para realizar os testes e o processo de depuração interativa ("try-out").

Como primeiro passo a ser realizado o programa deve ser pré-processado. Este procedimento é realizado no módulo Pré-processador e é feito em duas etapas. Na primeira o programa é interpretado para construir o plano de trajetórias, para posteriormente realizar os processos de interpolação para cada uma das trajetórias. Uma vez que o pré-processador já calculou cada uma das coordenadas (no espaço de juntas) para o programa em questão, este pode ser validado visualmente através da tela usando o módulo Simulador Gráfico.

Estas simulações podem incorporar a representação dos elementos adicionais que formarão parte da aplicação. Para este objetivo existe o módulo Gerenciador de "Lay-outs". Nele o usuário pode criar, modificar e alocar objetos e/ou conjuntos de objetos, chamados de "Lay-outs", com o objetivo posterior de alcançar uma representação mais real da aplicação, do manipulador e da disposição do envelope de trabalho.

Além disso, o usuário poderá detetar as possíveis colisões que poderão acontecer entre o manipulador e outros objetos do "Lay-out". Assim cada configuração do envelope de trabalho

terá associada a ela um "Mapa de configuração", com as informações relacionadas aos espaços livres por onde o manipulador poderá transitar sem colidir com algum objeto, e os espaços proibidos (onde existirão colisões). O módulo de Detecção Automática de Colisões confrontará as informações do Mapa de Configuração com cada uma das posições que o manipulador irá ocupar durante a execução do programa. Dessa forma, o usuário poderá ter dados sobre as posições onde o manipulador estará colidindo com algum objeto do "lay-out" e posteriormente realizar as modificações pertinentes. Isto permite ao usuário possuir de mais uma ferramenta, além da visual, para testar a viabilidade e eficiência dos seus programas. A seguir serão explicados mais detalhadamente estes quatro módulos.

4.1.PRÉ-PROCESSADOR :

Como já foi dito, um dos objetivos principais na elaboração deste sistema é provar a factibilidade e eficiência dos programas robóticos escritos numa linguagem textual. Através da representação gráfica, o programador poderá testar a lógica usada nos algoritmos, além de comparar diversas configurações do espaço de trabalho e principalmente otimizar as trajetórias do manipulador em função dos requerimentos e restrições da aplicação específica.

A principal atribuição dessa representação é a alta confiabilidade e grau de realismo na visualização da execução do programa. Em outras palavras, o simulador deve mostrar inequivocamente, o que o programador expressou através do algoritmo e pelo uso de uma linguagem de programação off-line.

Os diversos sistemas de simulação gráfica para programas escritos "off-line" usam dois tipos de aproximação para executar a simulação dos movimentos.

○ A criação de uma linguagem de interface, própria do simulador. Uma vez que a tarefa de programação foi completada, cumprindo-se todos os requerimentos da aplicação, o programa estará apto para a implantação real. Este programa fonte é, então traduzido para o formato da linguagem específica do controlador do robô real (Figura 4.3).

○ Uma técnica alternativa, consiste em programar-se o simulador na mesma linguagem do robô, evitando-se a tarefa de traduzir o programa em outra linguagem. Este método reduz ao mínimo os erros de tradução e permite ao programador depurar o programa interativamente numa linguagem que será mais natural para ele **[MILB-88]** (figura 4.4).

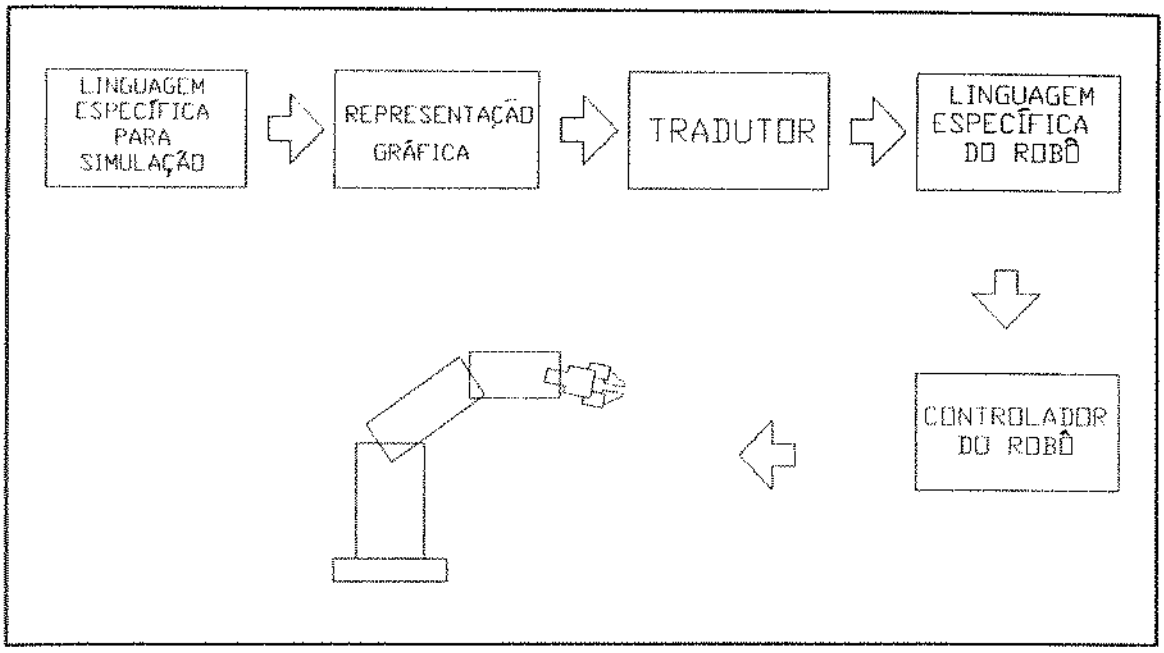


Fig. 4.3. Simulação através de uma linguagem de interface.

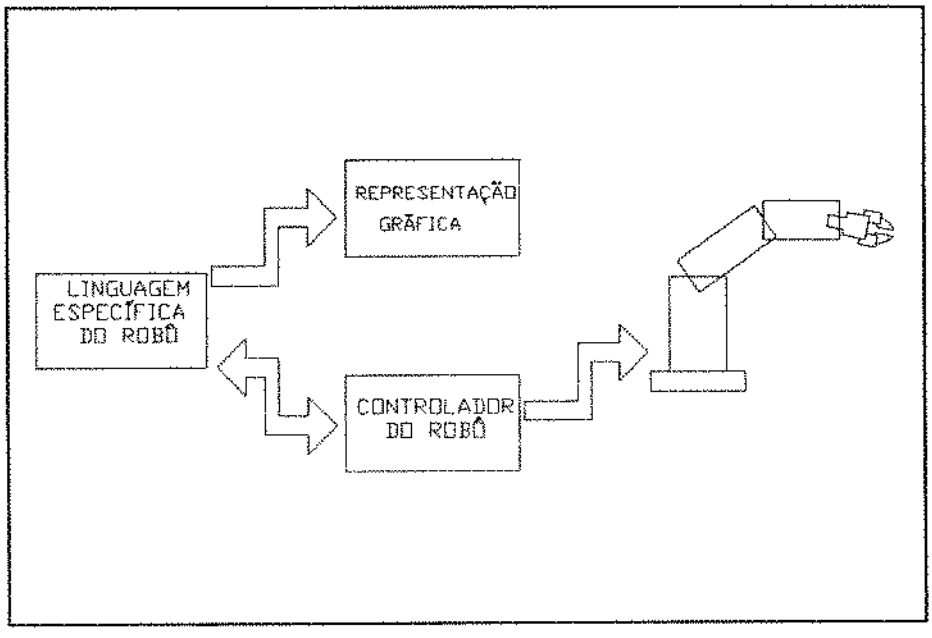


Fig.4.4 Simulação com a linguagem própria do robô.

Este trabalho adotou como base esta última técnica, principalmente para evitar a possibilidade de erro causado pelo fato de existir um eventual processo de tradução que poderá causar problemas no momento da implantação do programa no controlador do robô real.

Por outro lado, considerando a tendência atual para a padronização e modularidade, direcionada principalmente à construção de sistemas CIM, os diversos sub-sistemas devem adequar-se às necessidades específicas do usuário particular além de permitir ampla possibilidade de interfaceamento com outros subsistemas, inclusive provenientes de outros fabricantes ou desenvolvidos completa ou parcialmente "in-house" [TROS-88].

Para possibilitar a adequação do simulador gráfico a diferentes linguagens de programação textuais, existe a necessidade de definir um formato neutro de comunicação entre estas diferentes linguagens e o módulo de representação gráfica [COUR-81].

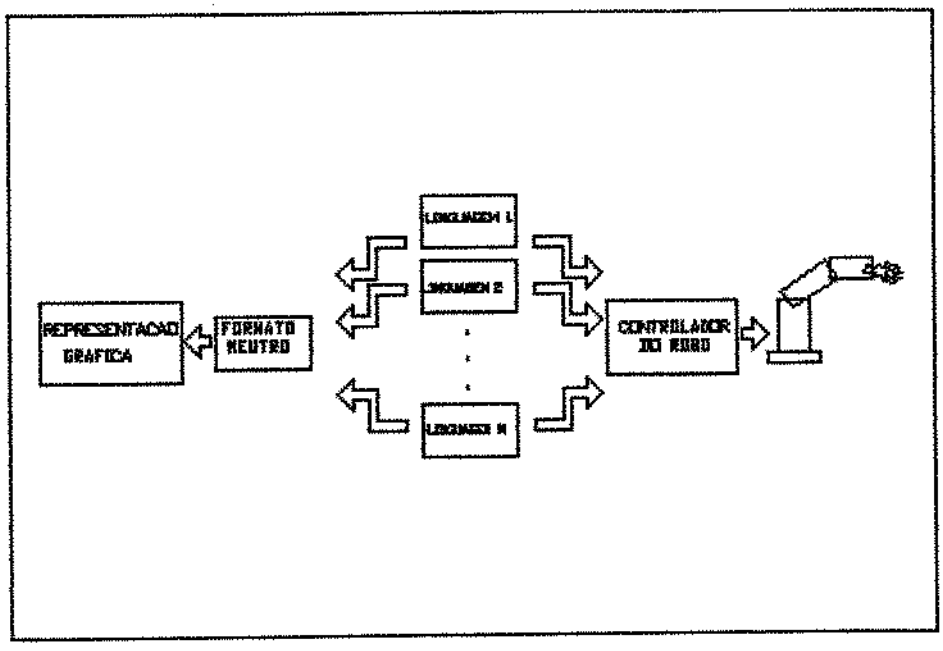


Fig. 4.5. Conceito de interface neutra na simulação.

A figura 4.5 esquematiza o conceito de Interfaces Neutras. Este tem sido usado em diferentes estágios na integração de diferentes módulos e sistemas de automação tanto para atividades de projeto, como de manufatura (CAD/CAM). Dois exemplos destacados deste

conceito são constituidos pelo padrão IGES, que constitui hoje em dia um dos formatos mais usados na transferência de informação gráfica para e desde pacotes CAD [MAYE-87], e o conjunto de especificações IRDATA que pretende padronizar através de uma linguagem de nivel médio a informação necessária para o controle de um manipulador industrial [WECK-87]. Através de uma interface neutra a informação é transferida desde o programa textual ao simulador gráfico, num formato comum e neutro. Assim, o trabalho de programação torna-se completamente independente de qualquer linguagem específica. Aliás, através deste método evita-se a criação e aprendizado de uma linguagem específica para o simulador por parte do programador.

A transferência é realizada à partir de um arquivo nativo, neste caso um programa escrito numa linguagem textual específica, para um arquivo neutro, cujo formato é entendido pelo sistema de representação gráfica. Esta operação é feita através de um software chamado pré-processador [EUSE-84].

Os pré-processadores constituem a única parte dependente da linguagem de programação que está sendo usado para definir as tarefas do manipulador, e deve ser construído um para cada linguagem que se deseje simular. Porém, ao se analisar as linguagens de programação textual existentes hoje em dia, conclui-se que suas estruturas e comandos possuem grande semelhança. Isto implicará na possibilidade de diversos pré-processadores compartilharem uma mesma estrutura principal e possivelmente alguns deles terão relativamente pequenas modificações.

A seguir são apresentados os principais conceitos da estrutura do pré-processador construído para programas escritos na linguagem AML.

Este sistema está composto por dois programas principais, o interpretador que realiza a partir do programa em AML a construção da tabela de trajetórias, e um interpolador que, usando como entrada, a tabela de trajetórias definidas pelo programa anterior, calcula todos os pontos intermediários para cada uma dessas trajetórias. O seguinte esquema representa o processo completo (figura 4.6).

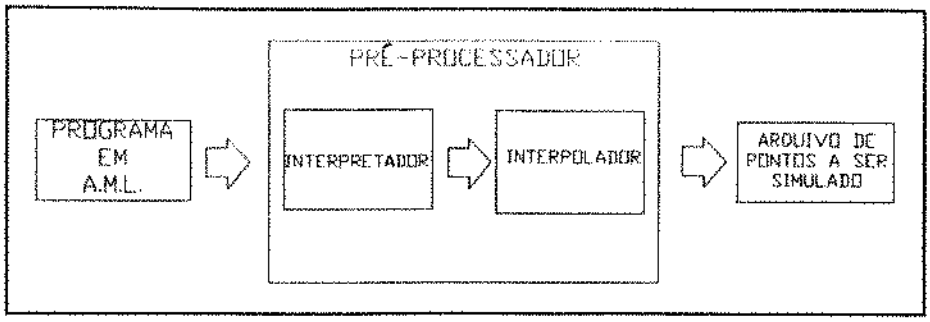


Fig.4.6. Esquema do processo de pré-processamento.

Analizaremos com mais detalhes ambos os programas que fazem parte do módulo Pré-processador.

4.1.1. INTERPRETADOR

O objetivo deste sub-sistema é, a partir de um programa de aplicação escrito na linguagem textual AML, gerar uma lista com as trajetórias definidas, junto com um conjunto de parâmetros que influenciam o tipo de movimento **[GOLD-84]**. Esta tabela tem a seguinte forma:

Tabela 4.1 Lista de Trajetórias de um programa.

Ponto Origem				Ponto Destino				Tipo Movimento			Garra
X	Y	Roll	Z	X	Y	Roll	Z	Linear	Payload	Zone	G/R

Onde X e Y são as coordenadas cartesianas no plano horizontal dos pontos Origem e Destino. R representa o ângulo Roll da garra, tanto para a configuração de partida como para a meta. Z especifica a posição vertical do terceiro elo. Esta posição somente pode assumir dois valores: 'U' se o elo está na posição superior e 'D' se está na posição inferior.

Os campos LINEAR, PAYLOAD e ZONE[3] armazenam os valores dos parâmetros de velocidade, tipo de movimento e precisão de posicionamento ativos para a trajetória correspondente.

Finalmente a coluna 'GARRA' especifica o estado (aberto/fechado) do atuador. A ação de abrir ou fechar a garra é considerado como uma trajetória portanto, cada vez que exista este tipo de ação, um novo registro será gerado nesta tabela. Este registro terá as coordenadas de origem e destino idênticas e o único campo que sofrerá alteração será o campo de GARRA.

O processo de construção dessa tabela divide-se em duas partes. A primeira consiste na definição de várias listas que conterão todas as variáveis definidas pelo programador ao interior do programa. Estas listas estão agrupadas dependendo do tipo da variável definida e além de armazenar o nome de cada uma delas, mantêm tanto o valor inicial como o valor instantâneo de cada uma delas ao longo da execução do programa.

A primeira destas listas é chamada de VALCONT.TAB e possue a seguinte estrutura (Fig. 4.7.)

VALCONT.TAB	
NOME	TIPO
NomeValCont	String
ValorValue	Real

Fig. 4.7. Estrutura da lista de Values e Contadores.

Nesta lista serão armazenados os "values" e contadores definidos no programa. O interpretador manipula ambos os tipos da mesma forma, com a única diferença que o valor dos contadores pode ser alterado ao longo da execução do programa, à diferenca dos "values" que

[3] LINEAR, PAYLOAD e ZONE são primitivas da linguagem AML que podem ser elucidadas no Apêndice dessa dissertação.

mantêm o seu valor constante durante toda a simulação.

A segunda lista a ser construída é a lista dos "aggregates"; nela são armazenados todos os conjuntos definidos, identificando cada um dos seus componentes, o tipo e se for necessário o valor adotado para cada um deles. A estrutura desta lista é a seguinte (Figura 4.8.)

AGGREGATE_TAB	
NOME	TIPO
NomeAggregate	String
TipoAggregate	Integer
NomeElemento	String
ValorAggregate	Real

Fig. 4.8. Estrutura da lista de Aggregates.

No campo NomeAggregate serão armazenados os nome de cada um dos "aggregates". O segundo campo identificará o tipo de elementos que estão contidos no "aggregate". Assim, este campo pode tomar os seguintes valores :

0 para "aggregate" de Números
1 para "aggregate" de Pontos
2 para "aggregate" de Contadores e Values.

O terceiro campo contém o nome de cada um dos elementos pertencentes ao "aggregate"; este campo só será usado se o "aggregate" for um conjunto de pontos, "values" ou contadores. Finalmente o último campo conterá o valor numérico, de cada um dos elementos do "aggregate", se o tipo especificado for 0 (Aggregate de Números).

A terceira lista contém todos os pontos definidos pelo programador para serem usados de maneira simbólica ao longo do programa. Esta lista tem o seguinte formato (Figura 4.9):

PONTOS_TAB	
NOME	TIPO
NomePonto	String
X	Real
Y	Real
Roll	Real

Fig. 4.9. Lista de Pontos.

O primeiro campo contém o nome dado pelo programador para identificar a localização no espaço de trabalho. Os Campos X e Y armazenam as coordenadas cartesianas respectivas. O último campo, o roll, conterá o ângulo da garra em torno ao eixo vertical Z.

Existem outras duas listas que são construídas durante este processo, que diferentemente das anteriores, não armazenam informações referentes às variáveis do programa. Estas duas tabelas contém a informação que guiará o controle do fluxo durante a interpretação dos comandos. A primeira dessas tabelas e de Subrotinas, que possue a seguinte estrutura (Fig. 4.10):

SUBR_TAB	
NOME	TIPO
NomeSubr	String
LocalSubr	Integer
NomePtoSub	String

Fig. 4.10. Tabela que armazena informação das subrotinas.

No campo NomeSubr armazena-se o nome de cada subrotina contida no programa.

LocalSubr contém o ponteiro indicando a localização dentro do programa, onde começa cada subrotina. No último campo, NomePtoSub, serão guardados os nomes de cada um dos parâmetros locais, se existem relativos à subrotina dada.

Assim, toda vez que o pré-processador encontra uma chamada de uma subrotina, será consultada esta lista, para ler o ponteiro e direcionar a transferência do controle de fluxo. Se a chamada contém parâmetros, estos parâmetros locais - contidos na tabela no campo NomePtoSubr - serão carregados com os valores adequados, contidos na chamada.

A última lista armazena todas as etiquetas (LABELS) que existem dentro do programa junto com um ponteiro indicando a localização delas ao interior da seqüência de comandos. De tal maneira, cada vez que o pré-processador encontrar um comando de desvio (condicional ou não condicional) irá transferir o controle de fluxo à nova direção, para continuar com o processo de interpretação. A estrutura dessa lista é a seguinte (Figura.4.11):

ETIQ TAB	
NOME	TIPO
NomeLabel	String
LocLabel	Integer

Fig. 4.11. Lista de LABELS.

A segunda parte do processo de construção da tabela de trajetórias, geralmente é chamada de "Parsing" **[KERN-76]**. Nele posiciona-se um ponteiro na primeira linha de instruções do programa principal. Depois inicia-se um processo de varredura (rastreio) ao longo da seqüência de comandos. As instruções são lidas uma a uma e, depois de ser identificadas, transfere-se o controle para um procedimento específico para cada instrução.

Este procedimento específico gerará as ações adequadas ao comando AML sob análise. Uma vez que todas as ações têm sido realizadas, o ponteiro de leitura é incrementado repetindo o processo de interpretação para a instrução seguinte **[SZPA-87]**.

Note que este processo não realiza uma revisão da sintaxe do programa, já que supõe

que este está correto sintaticamente. O que ele faz apenas, é construir uma representação simbólica das ações do manipulador definidas no programa textual em AML.

Duas características especiais da linguagem AML merecem um tratamento especial na construção do pré-processador. A primeira refere-se à manipulação da informação sensorial e a segunda está relacionada ao recurso de programação avançado através de subrotinas.

A linguagem AML tem uma importante capacidade de manipulação de informação sensorial, que permite-lhe realizar (programar) desvios condicionais, dependendo do estado de determinados transdutores. Durante a simulação isto é resolvido através da interação como usuário. Quando o interpretador acha um comando deste tipo, emite um "prompt" ao usuário, perguntando-lhe se a condição programada deve ser considerada como verdadeira ou falsa. Dependendo da resposta a esta pergunta, será a direção de fluxo do processo de interpretação. Isto permite ao usuário visualizar diversas alternativas, principalmente em relação ao estado de dispositivos existentes na aplicação específica.

Uma das características principais da linguagem AML é a sua orientação a subrotinas. O uso desse recurso é frequente no desenvolvimento de programas de aplicação. Uma subrotina é um subprograma que realiza uma operação específica e que é usada em várias ocasiones durante a execução de um programa. Para a manipulação destes subprogramas o interpretador executa um mecanismo recursivo; isto é, cada vez que o interpretador achar uma chamada a subrotina, o corpo dessa subrotina, é tratado como se fosse um programa em si, iniciando a busca seqüencial desde a primeira linha de instruções, até achar um "END;", que indicará o final da subrotina. Quando isto acontecer o controle de fluxo voltará ao programa principal ou à subrotina desde a qual originou-se a chamada.

Uma vez tendo sido completado o processo de interpretação do programa, ativa-se o segundo processo, a interpolação. Este processo será analisado com maior detalhe no seguinte ponto.

4.1.2. INTERPOLADOR

O segundo programa do módulo de pré-processamento está dedicado à interpolação das trajetórias existentes na tabela gerada pelo interpretador. Como já foi dito, além das coordenadas dos pontos de origem e meta, esta também contém informação referente ao tipo de movimento que há de ser realizado. Esta informação governará os processos de interpolação necessários para cada movimento.

Utilizando-se a linguagem AML é possível programar-se movimentos segundo dois tipos de trajetórias **[GROO-89]**.

O primeiro é o chamado de trajetórias no espaço das juntas, ou método de interpolação de juntas, onde o controlador calcula a quantidade de tempo que levará para alcançar seu destino à velocidade comandada para cada junta. Ele então escolhe o tempo máximo entre estes, e usa esse valor como tempo de referência para todos os eixos. Isto significa que uma velocidade separada é calculada para cada eixo. Este tipo de trajetórias é inicializado na linguagem AML através do comando LINEAR (0) (Valor Default).

Por outro lado, às vezes é requerido um tipo de trajetória linear para realizar tarefas específicas. Nesse caso utiliza-se o segundo método de interpolação, chamado interpolação linear, onde o orgão terminal se desloca descrevendo uma trajetória reta definida em coordenadas cartesianas **[WANG-89]**.

A saída deste programa consiste em uma tabela contendo os ângulos de cada uma das duas primeiras juntas ($\Theta 1, \Theta 2$), além do ângulo ROLL da garra. Cada trajetória tem associada a ela dez pontos, correspondendo o primeiro e o último aos pontos origem e destino respectivamente. Os oito pontos restantes são os pontos intermediários ou "via-points".

Esta tabela é armazenada em um arquivo do tipo ASCII, para sua posterior representação gráfica. O fato desta tabela estar no formato ASCII, permite ao usuário criar ou modificar interativamente esse arquivo através de qualquer editor de texto que trabalhe nesse padrão.

4.2. GERENCIADOR DE "LAY-OUTS"

Para cada programa ou aplicação que o usuário deseje simular, ele tem a possibilidade de definir objetos e configurações para o espaço ou célula de trabalho. Assim, existe um módulo onde podem ser criados, modificados e alocados objetos e/ou conjuntos de objetos, chamados de "lay-outs", com o objetivo posterior de serem incorporados à simulação gráfica de um programa.

Este módulo está formado por dois sub-módulos, o primeiro realiza a tarefa de definição e gerenciamento de objetos, enquanto o segundo realiza as tarefas de integração entre os objetos definidos, para formar uma configuração do espaço de trabalho ou "lay-out".

Chama-se objeto a um paralelepípedo com dimensões e orientação definidas, no espaço cartesiano, com respeito a um sistema referencial equivalente ao usado como referencial absoluto na representação do robô.

A informação necessária na definição de um objeto está baseada nos seguintes parâmetros:

- Largura
- Comprimento
- Altura
- Rotação segundo os três eixos do sistema cartesiano.

Onde a altura do objeto é definida ao longo do eixo Z, o comprimento ao longo do eixo X e a largura ao longo do eixo Y.

A rotação segundo os eixos cartesianos não é armazenada de maneira explícita, se não que, cada vez que o usuário define um novo objeto, ele tem a possibilidade de realizar rotações independentes segundo qualquer um dos três eixos. Então, cada rotação realizada pelo usuário irá alterar os dados da matriz que contém a informação referente aos vértices e ficará fixa até o usuário realizar uma outra operação de rotação sobre o objeto em processo de definição.

Isto permite ao usuário definir várias orientações para um mesmo objeto, dando uma maior flexibilidade na hora de definir "lay-outs" para uma aplicação dada.

A seguir são explicados alguns aspectos importantes da operação do sub-módulo de gerenciamento de objetos. Uma vez que o usuário tenha entrado nesse módulo, o menú principal apresenta todas as possibilidades que ele terá para a definição e edição dos objetos (figura 4.12).

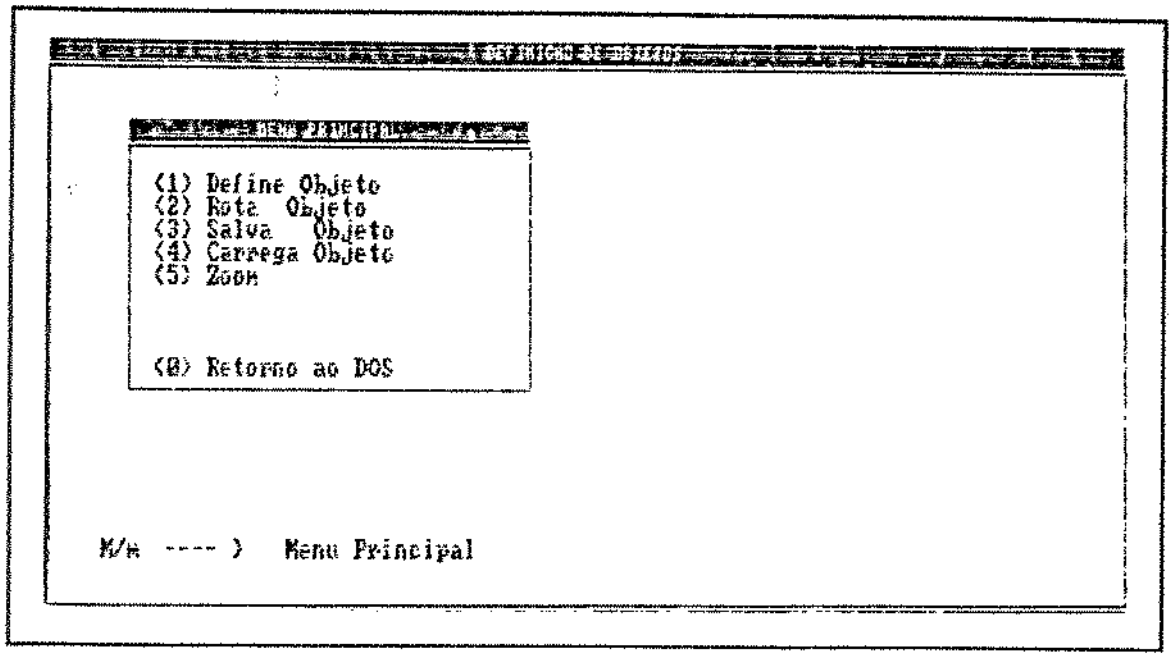


Fig.4.12. Menú principal do gerenciador de objetos.

Para definir um novo objeto (opção <1> do menú principal) o usuário é requerido pelas dimensões gerais do objeto segundo os três eixos de referência (figura 4.13).

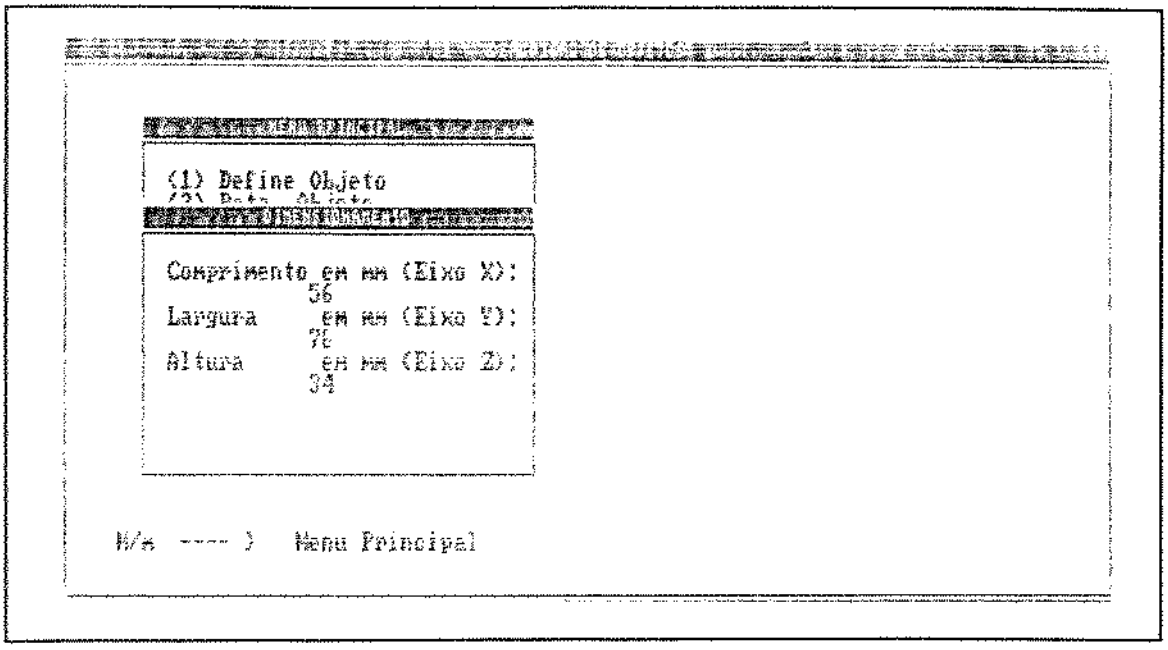


Fig.4.13. Entrada das dimensões do objeto.

Na figura 4.14. é apresentado o objeto definido pelas dimensões entradas pelo usuário através da tela mostrada na figura 4.6.

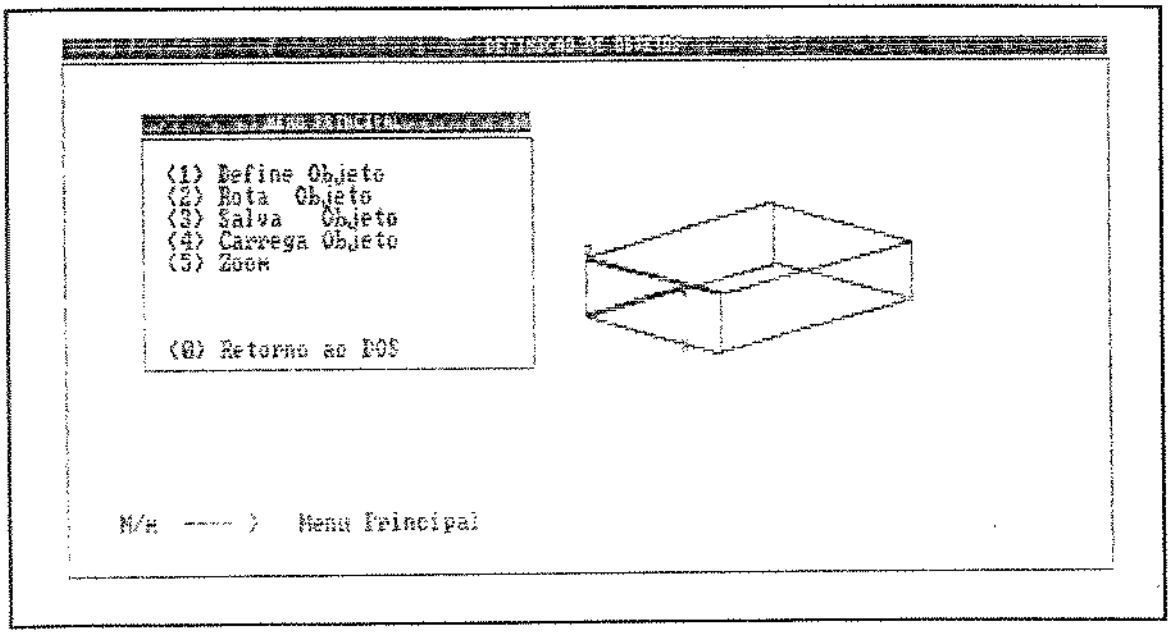


Fig.4.14. Representação do objeto definido

Adicionalmente o usuário pode precisar de fornecer uma orientação especial ao objeto definido. Isso é feito através da opção <2> do Menú Principal. Assim, o usuário definirá cada um dos ângulos que o objeto irá rotacionar em função de cada um dos três eixos de referência. As rotações podem ser feitas de forma independente, ou seja em torno de um eixo, mantendo-se fixas as orientações para os dois outros eixos, ou para todos os eixos simultaneamente. A figura 4.15. mostra a tela de definição das rotações para um objeto.

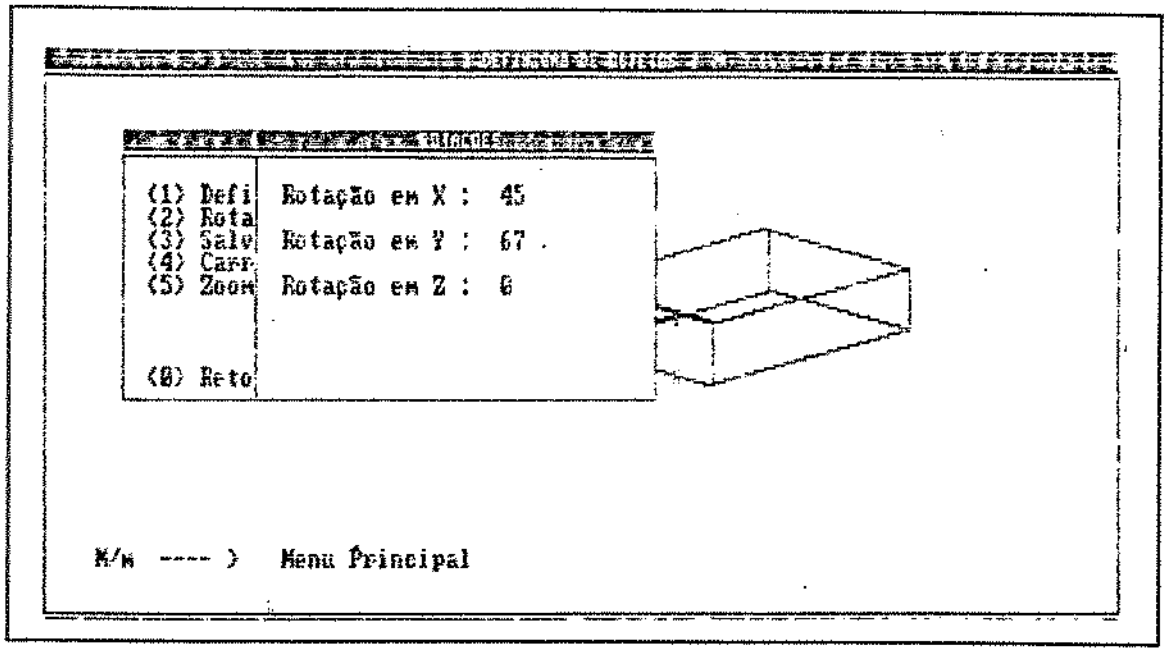


Fig.4.15. Tela de definição das rotações para o objeto

Note-se que o objeto será re-orientado segundo um ângulo de 45º em torno do eixo X, 45º em torno do eixo Y, e 12º em torno do eixo Z. A figura 4.16. mostra a duas orientações para o mesmo objeto, a orientação inicial, e a orientação resultante segundo os três ângulos definidos pelo usuário.

Um último recurso adicional, é o recurso de ZOOM, por meio do qual o usuário pode se aproximar o se afastar do objeto de maneira de observar de melhor forma algum detalhe do objeto em definição. Na figura 4.17. são apresentadas duas visualizações com dois fatores de Zoom diferentes para um mesmo objeto.

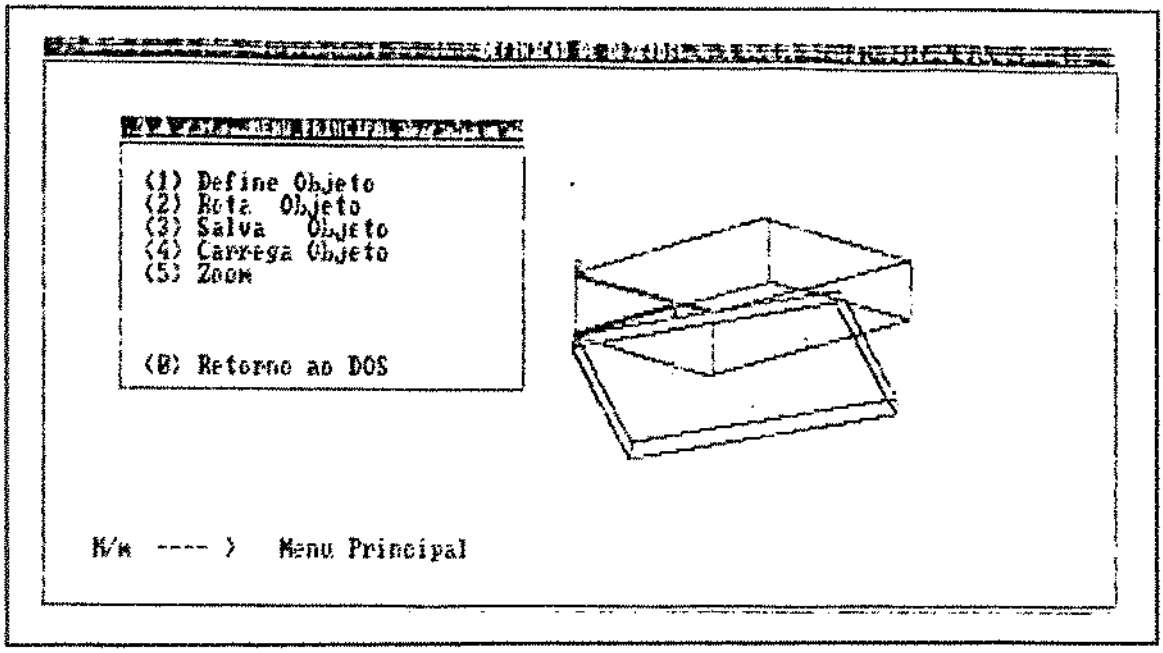


Fig.4.16. Representação da rotação de um objeto.

Uma vez, que o usuário tenha definido o objeto, ele deve armazenar esse novo elemento na base de dados para a sua posterior integração a uma nova configuração do espaço de trabalho. Na figura 4.18. é mostrado o processo de armazenamento de um objeto chamado "OBJECT1".

Posteriormente o usuário pode integrar diferentes objetos em uma determinada configuração o "lay-out" do espaço de trabalho permitindo representar (esquematicamente) diferentes dispositivos e/ou elementos que estarão presentes na aplicação real. Esta integração é feita no sub-módulo "Gerenciador de Ambientes" cujo menú principal é mostrado na figura 4.19.

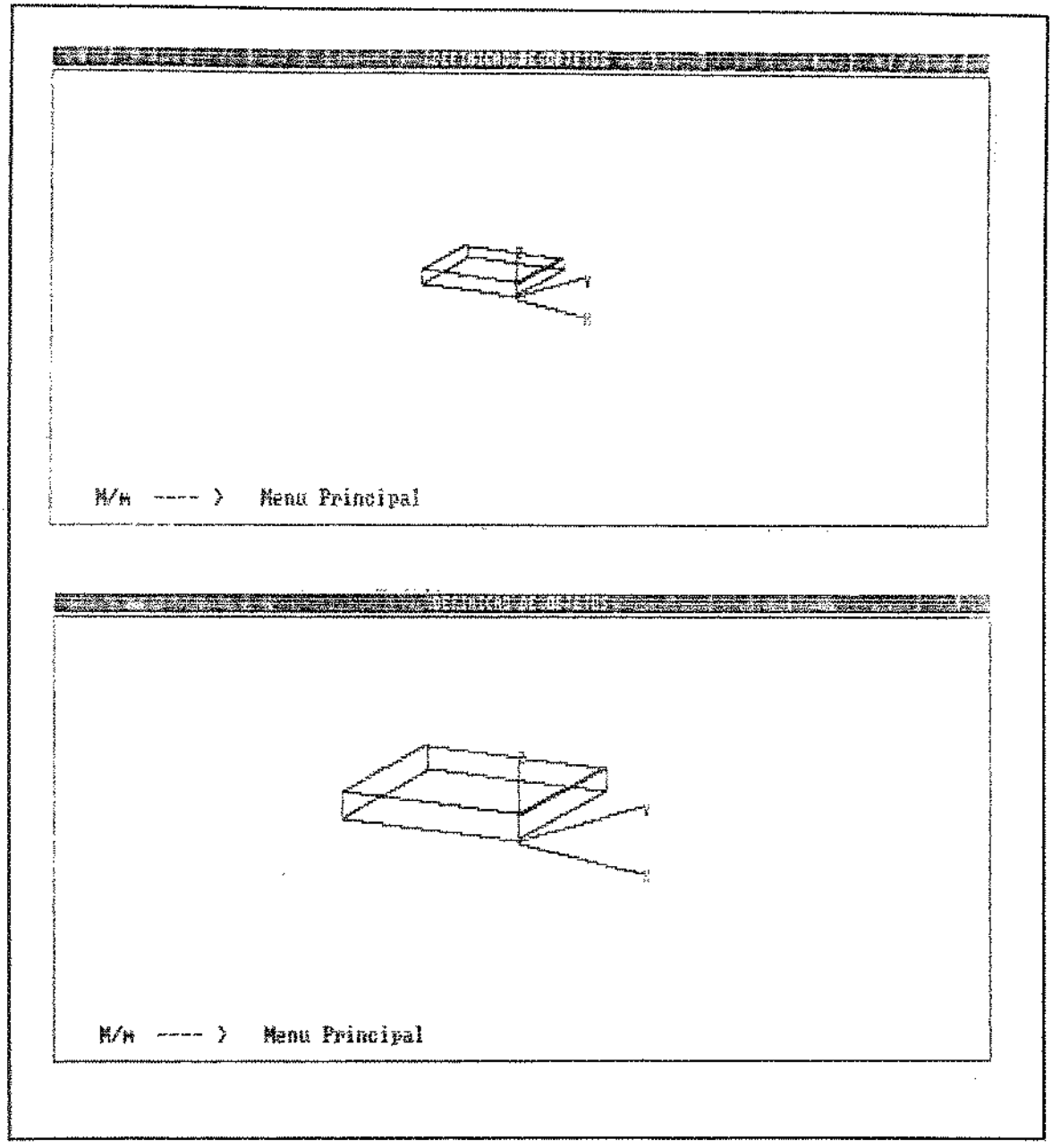


Fig. 4.17. Visualização do efeito "ZOOM".

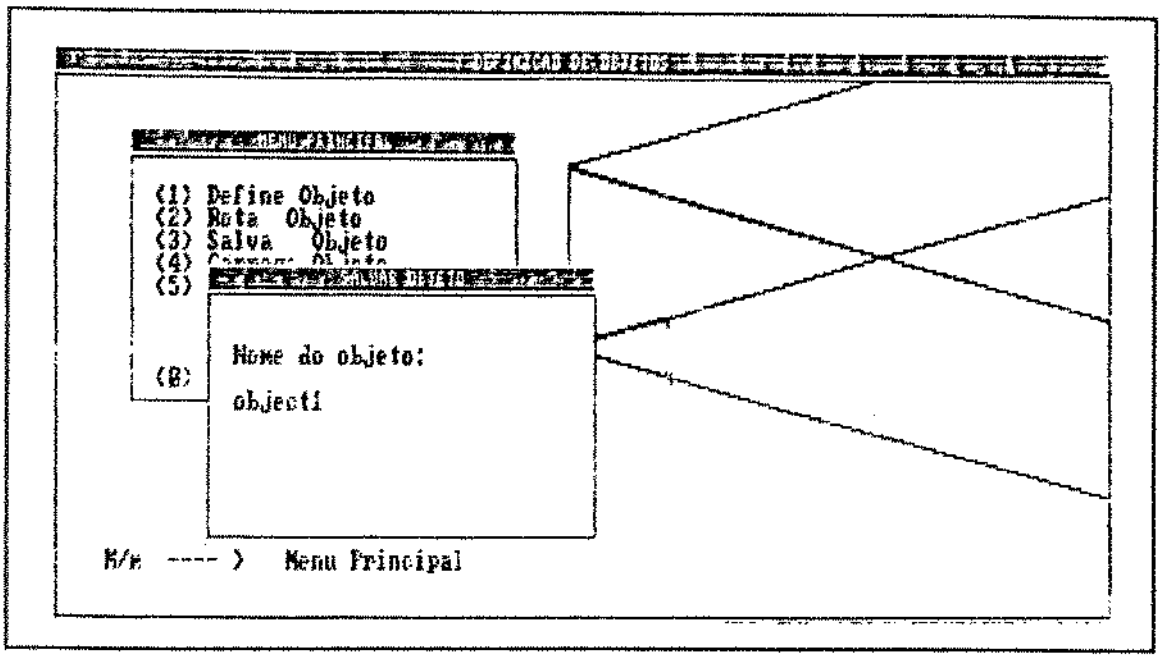


Fig.4.18. Armazenamento de um objeto definido

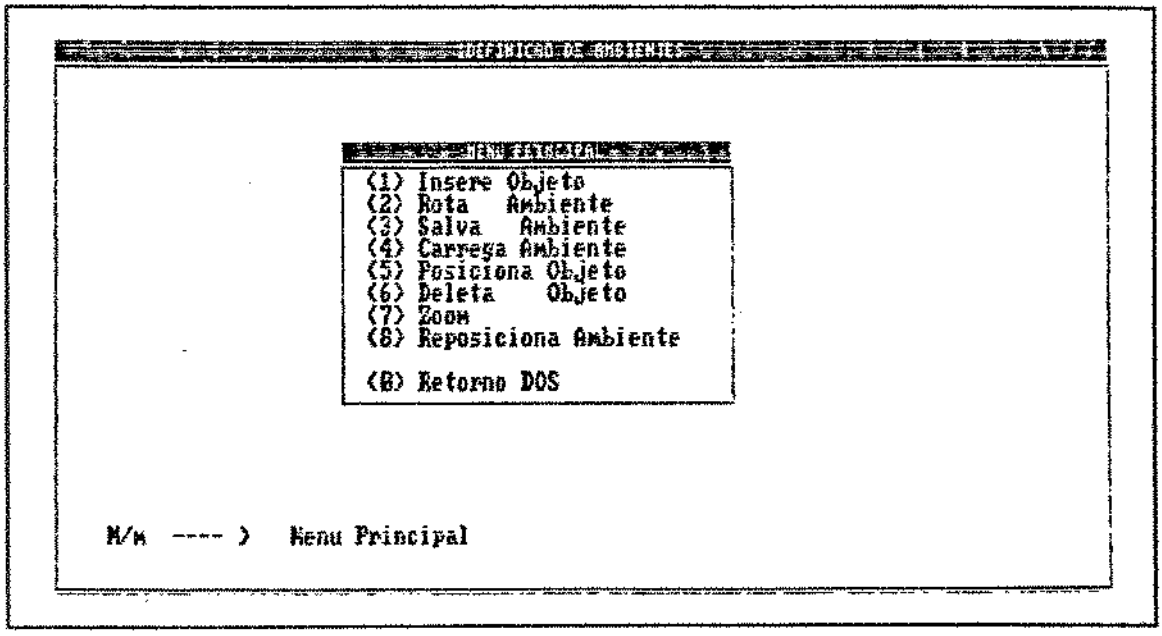


Fig. 4.19. Menú principal do gerenciador de ambientes

Para começar a definição de um novo "Lay-out" o usuário deve identificar cada um dos

objetos que estarão presentes na configuração do espaço de trabalho. Este objetos devem ter sido previamente definidos no módulo gerenciador de objetos. Através da opção <1> do menú principal (Fig.4.19.) o usuário identificará e localizará o objeto que deseja incorporar ao ambiente. A identificação é feita através do nome que o objeto recebeu quando ele foi definido e armazenado na base de dados e de um número de componente dentro do "lay-out"(figura 4.20.)

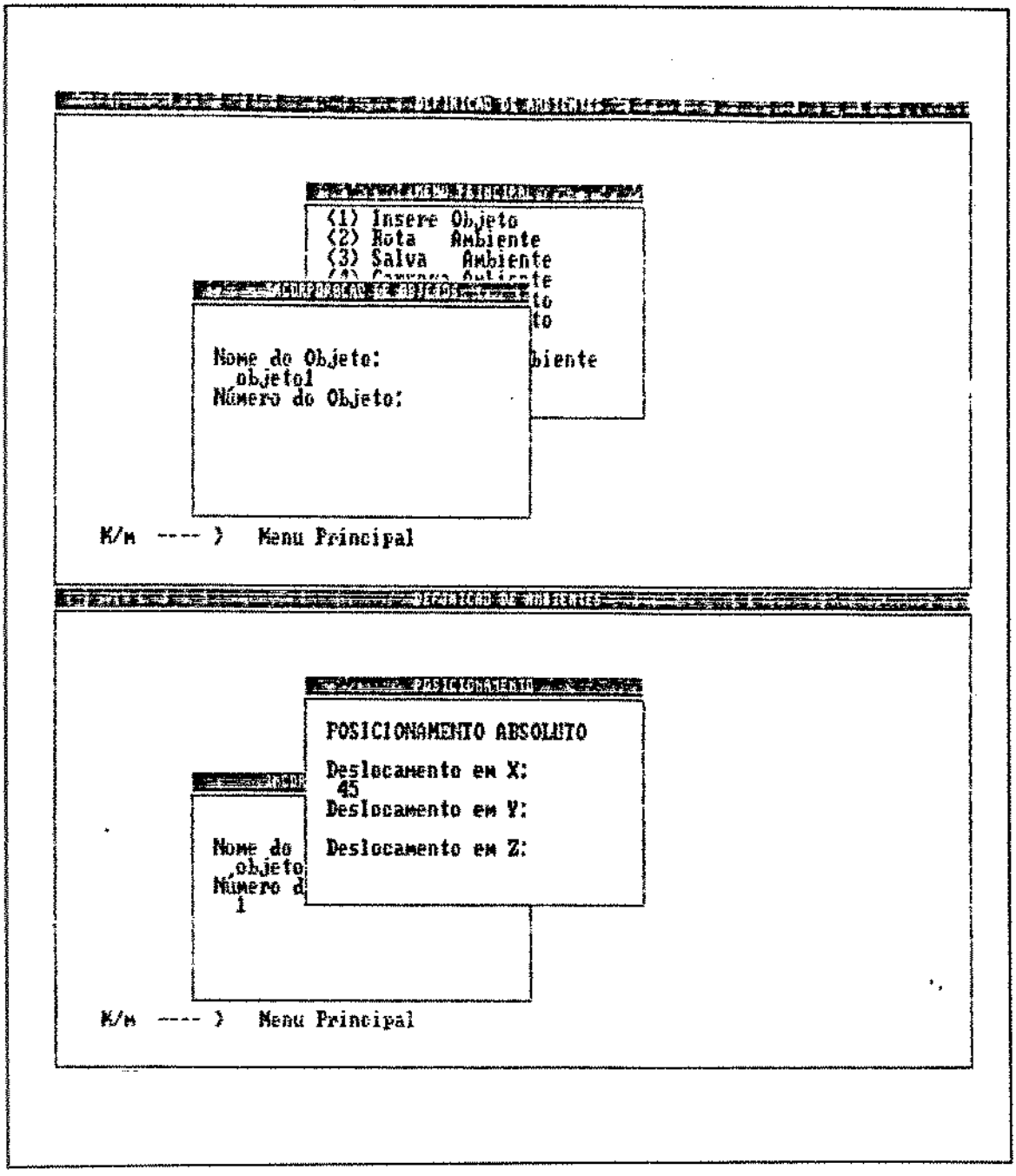


Fig. 4.20. Incorporação e localização de um objeto no "lay-out"

Atualmente o sistema pode armazenar e representar graficamente até oito objetos simultaneamente num mesmo 'lay-out', pudendo armazenar um ilimitado número de configurações, apenas restrito ao espaço disponível em disco. A figura 4.21. mostra um exemplo de "lay-out".

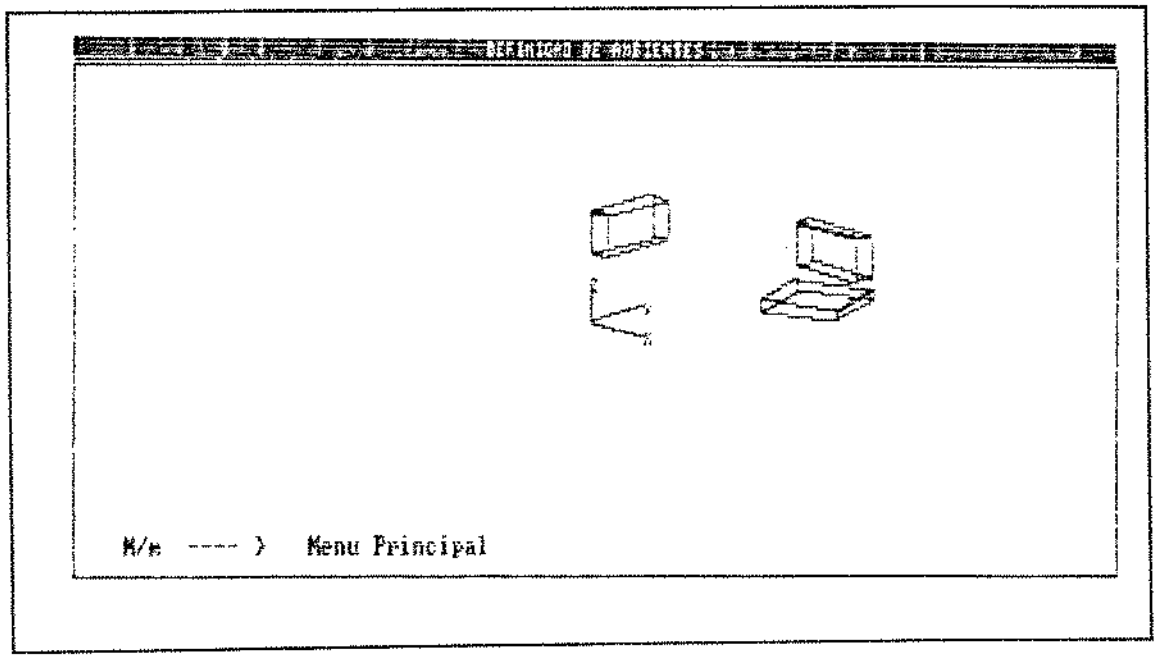


Fig. 4.21. Representação na tela de um "lay-out".

Neste sub-módulo o usuário pode alterar a posição de qualquer dos objetos presentes a través da opção <5> do menú principal, e modificar a orientação com respeito ao eixo vertical (Z) de toda a configuração. Na figura 4.22 é mostrado o processo de rotação do ambiente completo em um ângulo de 45º em relação ao eixo Z.

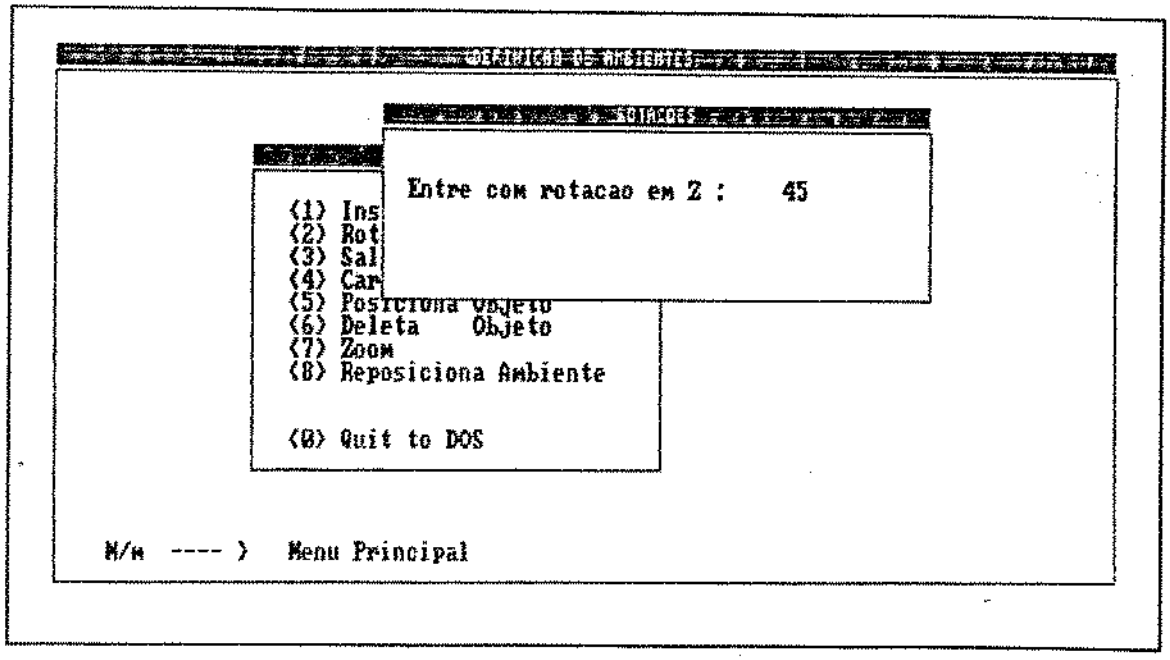


Fig. 4.22. Processo de rotação de um "lay-out" completo

O resultado desta rotação é mostrado na figura 4.23.

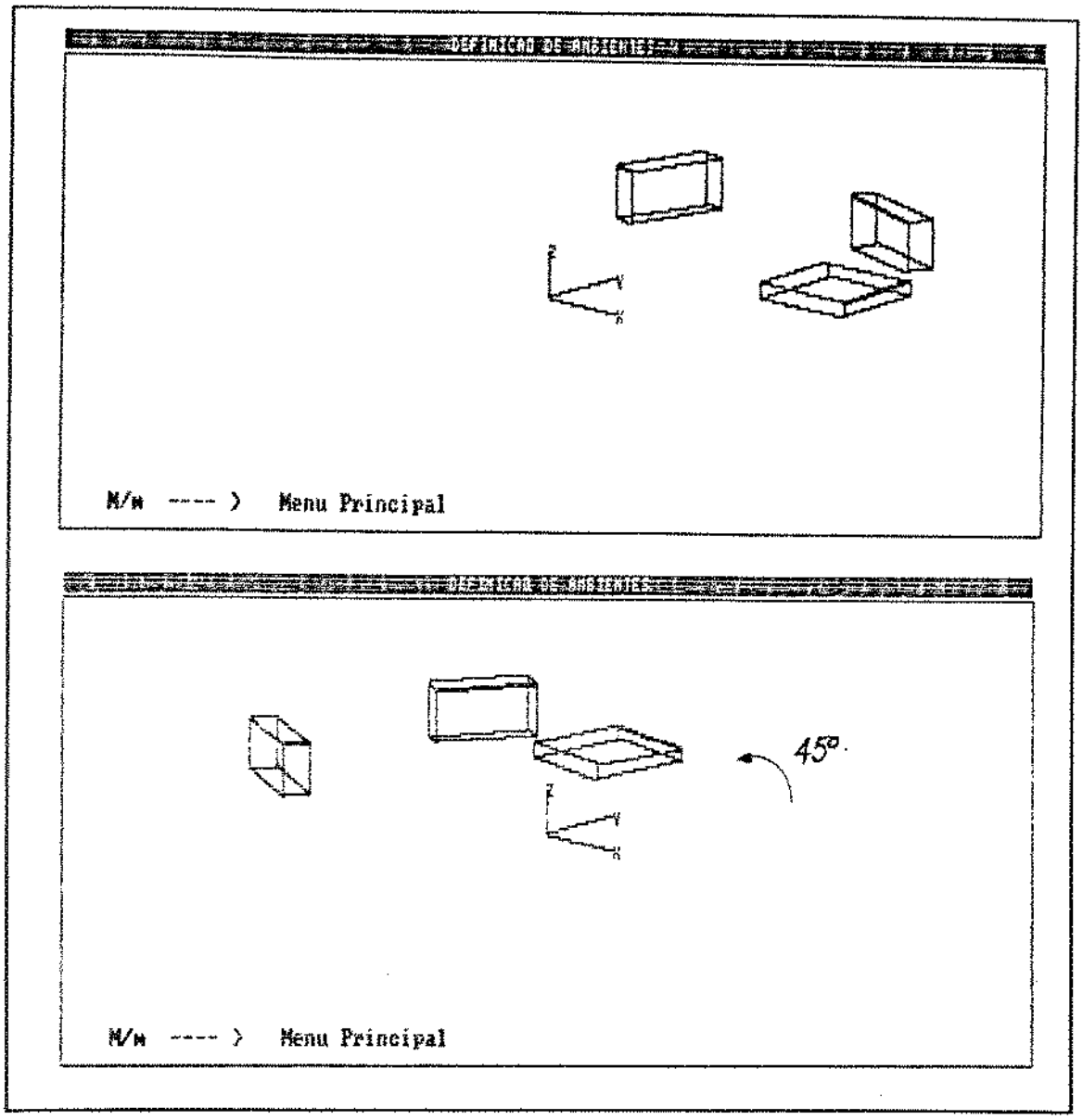


Fig. 4.23. Processo de rotação de um "lay-out" em 45o

Além da possibilidade de visualizar estes objetos e a posição relativa com o manipulador, no momento da simulação gráfica estes arquivos serão de vital importância num estágio posterior da simulação, isto é, em base a eles será desenvolvido o teste de deteção automática de colisões para um determinado programa de aplicação. Estes procedimentos serão discutidos no próximo ponto.

4.3. SIMULADOR GRÁFICO

Finalmente, uma vez que o usuário já definiu a configuração do espaço de trabalho requerida para uma certa aplicação, ele pode realizar a simulação do programa em AML já pré-processado pelo módulo interpretador e incorporar na representação os objetos que estarão presentes na aplicação real. Assim, obter-se-á uma visão mais real do processo e o usuário poderá se antecipar com maior precisão aos problemas que podem acontecer na execução do programa uma vez que este tenha sido implantado.

O usuário conta com seis possibilidades de visualização para os movimentos programados do robô. Para esse objeto, existe um menú gráfico interativo, por meio do qual o usuário pode escolher o tipo de representação que mais se-adecue às suas necessidades figura 4.24.

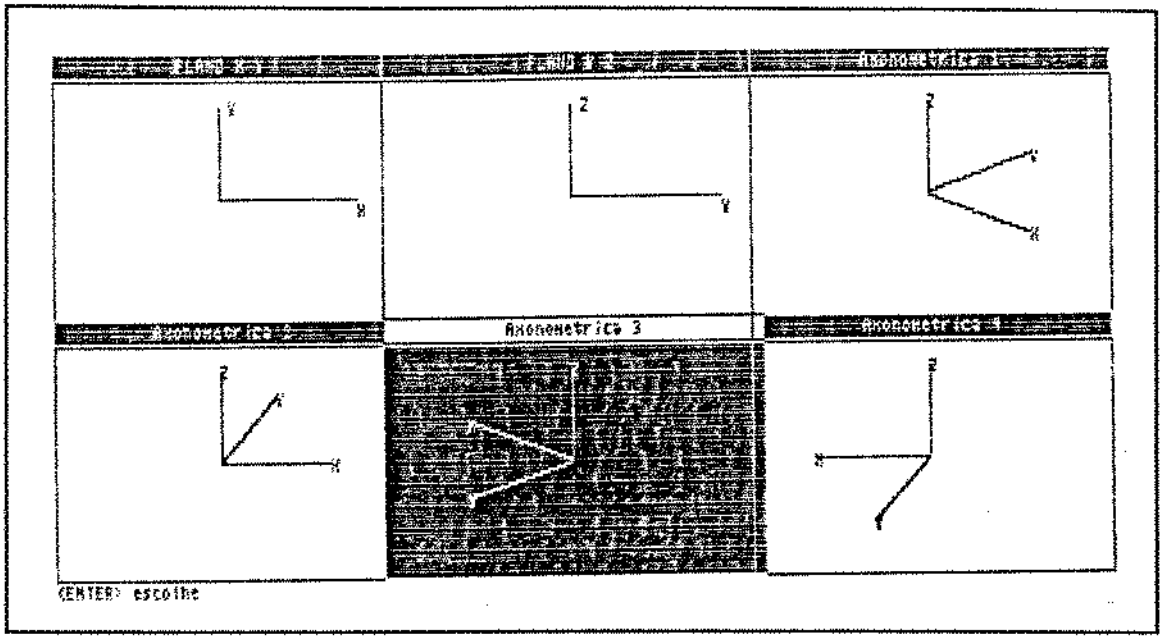


Fig. 4.24. Menú com as possibilidades de visualização

A primeira possibilidade é uma representação da planta do envelope de trabalho, onde podem ser observados os ângulos descritos pelas duas primeiras juntas e pela garra do manipulador figura 4.25.

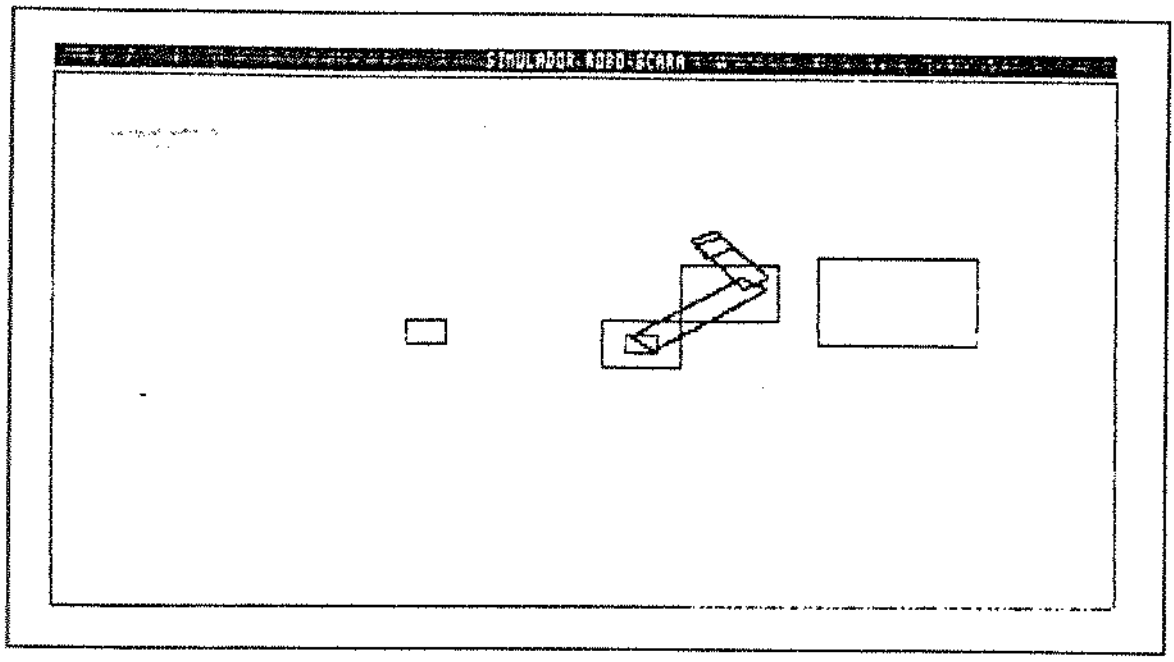


Fig. 4.25. Representação da planta do manipulador

A segunda possibilidade de representação também é de duas dimensões, e mostra um plano vertical do manipulador e o seu espaço de trabalho. Nela podem ser observadas principalmente as relações de altura entre os objetos existentes na configuração e os elos e garra do manipulador figura 4.26.

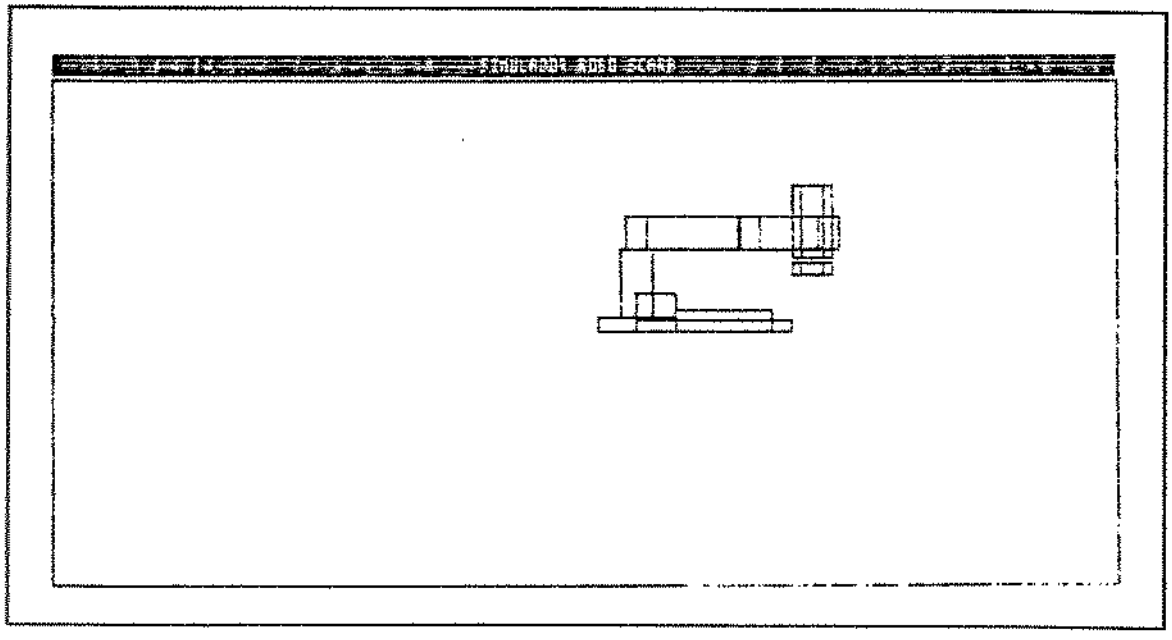


Fig. 4.26. Visualização em elevação do manipulador.

As quatro modalidades de visualização restantes são em três dimensões e representam em forma esquematizada (prismas regulares) o manipulador e os objetos existentes no espaço de trabalho. A diferença entre estas quatro possibilidades é função da posição do observador que varia em função do eixo vertical do sistema referencial global do robô. Assim o observador pode "navegar" ao redor do manipulador e o seu espaço de trabalho.

Uma vez que o usuário tenha definido qual o tipo de representação que deseja para desenvolver a simulação, ele será inquerido sobre se deseja ou não, incorporar a representação de um "lay-out" pré-definido à simulação gráfica (figura 4.27).

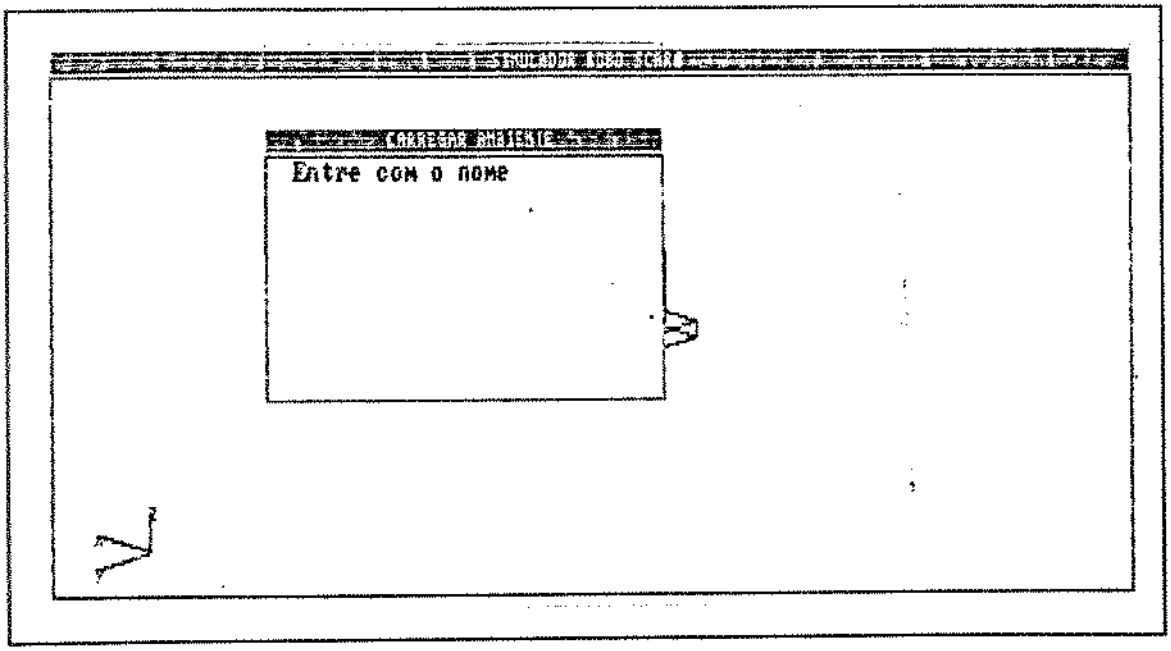


Fig. 4.27. Incorporação à simulação de um ambiente pré-definido

Na figura 4.28. mostra-se a representação do manipulador, na posição HOME, junto com o "lay-out" chamado "AMBIENTE2".

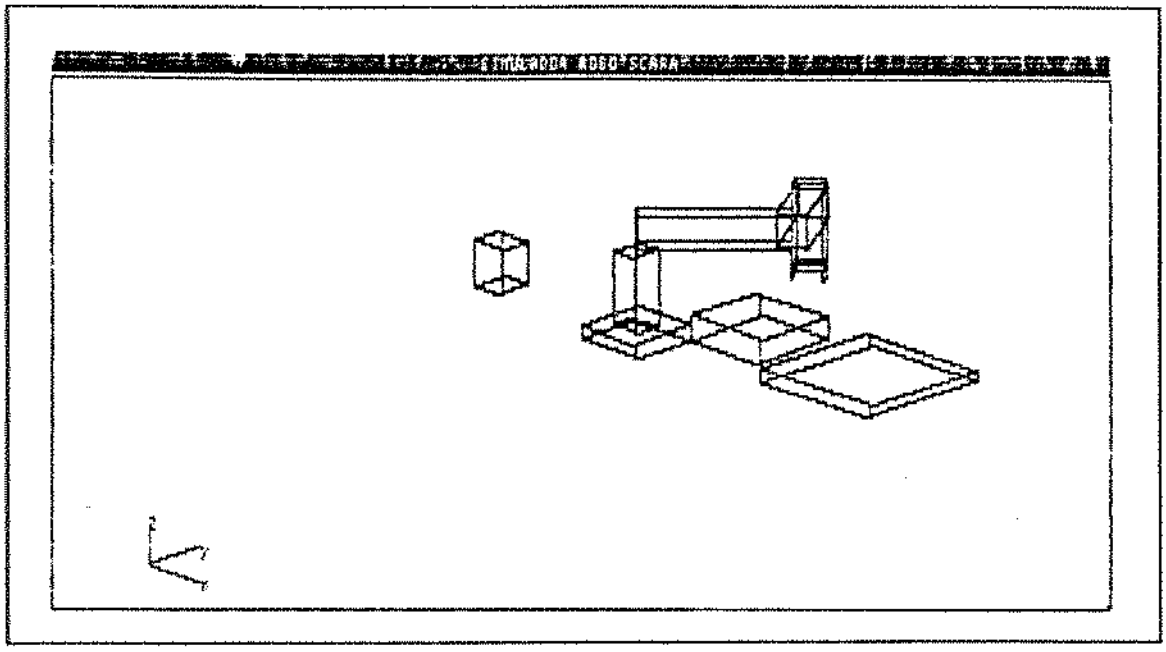


Fig. 4.28. Representação do manipulador e o ambiente

O processo de simulação gráfica está dividido em duas partes principais, na primeira parte são construidas, quadro a quadro, cada uma das visualizações que o manipulador descreve ao longo das trajetórias programadas. Cada um destes quadros será armazenado como um arquivo gráfico, do tipo "slide". Na segunda parte do processo de simulação o usuário pode acceder à animação gráfica, obtendo um efeito muito mais realístico dos movimentos do manipulador e a interação com o seu ambiente.

4.4. DETECÇÃO DE COLISÕES :-

Associado ao sistema de Gerenciamento de Lay-outs e ao módulo de simulação gráfica existe o módulo de detecção automática de colisões. Este módulo tem por objetivo detectar a existência de colisões entre as partes móveis do manipulador e os objetos definidos no Lay-out especificado. Esta detecção é feita automaticamente a partir do plano de trajetórias existentes na tabela de pontos gerada pelo processo de Interpretação. Este plano de trajetórias especifica cada um dos pontos, em coordenadas de junta, pelas quais o manipulador ira se deslocar. Cada uma destas coordenadas é comparada com o mapa de configuração, o qual armazena as informações de todas as coordenadas do espaço de trabalho que estão proibidas para o passo do manipulador, devido à existência de um objeto pré-definido.

A seguir são apresentados os conceitos mais importantes da definição e construção deste Mapa de Configuração além dos métodos usados para a detecção automática de colisões.

Lozano-Perez apresentou **[LOZA-83]** uma técnica para representação dos objetos existentes no espaço de trabalho, numa matriz, que ele chamou mapa de configuração. Este método basea-se na caracterização da posição e orientação de um objeto como um único ponto no espaço de configuração. Este espaço de configuração é uma matriz, na qual cada dimensão representa um grau de liberdade na orientação ou posição do objeto.

Assim, as configurações proibidas dentro desse espaço de configuração devido, à presença de outro objeto, podem ser representadas como uma região geométrica no mapa de configuração **[RED-85]**.

Considerando as características cinemáticas e geométricas do robô, e caracterizando o problema de detecção de colisões através de um método de pré-processamento e não em tempo real (durante a simulação), decidiu-se realizar uma adaptação destes mapas de configuração na representação das regiões proibidas do espaço de trabalho.

Este enfoque permite construir um mapa de configuração relacionado a cada um dos "lay-outs" do envelope de trabalho. Este mapa pode ser armazenado na forma de um arquivo de acesso direto (com o uso de ponteiros) e pode ser usado em múltiplos testes de avaliação com diversos planos de trajetórias.

Este método diminui bastante tempo computacional, já que, o mapa deve ser construído apenas no momento da definição do novo "lay-out".

Considerando que o terceiro elo do manipulador pode-se localizar segundo apenas duas posições (UP e DOWN), o mapa está baseado em duas matrizes bi-dimensionais. Assim, haverá uma matriz que especifica as posições proibidas do espaço de trabalho para quando o manipulador estiver na posição UP e outra matriz para a posição DOWN.

As coordenadas de cada uma destas matrizes estão baseadas nos ângulos das duas primeiras juntas (Θ1 e Θ2), como pode ser observado nas seguintes figuras (4.29. e 4.30.).

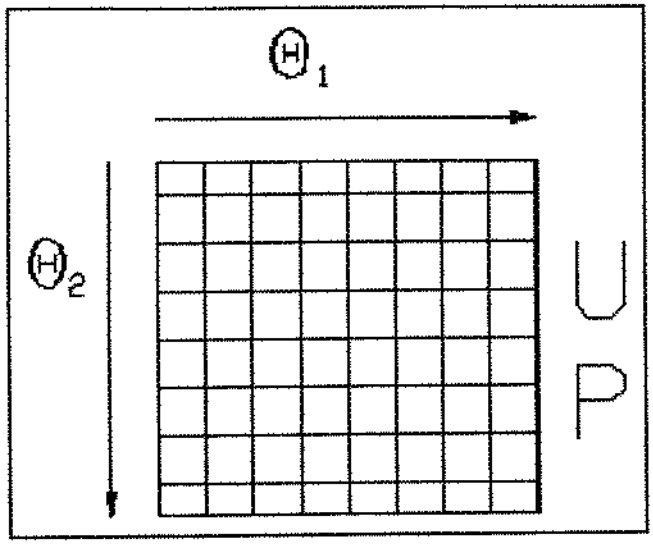


Fig. 4.29. Mapa de configuração para a posição UP

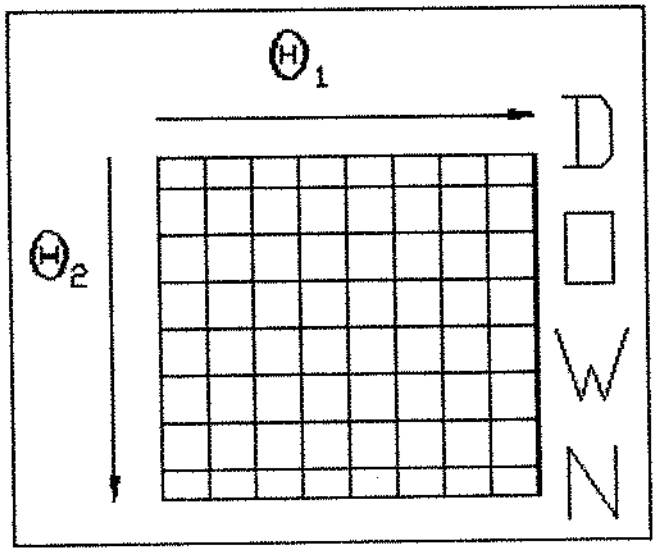


Fig. 4.30. Mapa de configuração para a posição DOWN

Para construir o mapa de configuração deve-se fazer primeiro o teste de colisões para o primeiro elo do manipulador. O espaço total de rotação de um elo que desloca-se planarmente pode ser representado pela seguinte equação:

$$\Delta\Theta_1(\gamma^1,\eta^1) = \Sigma^m_{i=1}\Delta\Theta_1{}^l_i(\gamma^l_i,\eta^l_i) + \Sigma^n_{j=1}\Delta\Theta_{1^p_j}(\gamma^p_j,\eta^p_j) \qquad (4.1)$$

onde $\Delta\Theta 1$ é o espaço total do elo 1, tendo como limites os ângulos γ^1,η^1. Este espaço corresponde à soma dos m espaços de configuração livres de colisão ($\Delta\Theta_i^l$, i = 1, ..., m) e dos n espaços de configuração proibidos ($\Delta\Theta_{1^p_j}$, i = 1, ..., n). Cada um desses espaços relaciona-se com os seus ângulos limites da seguinte forma :

$$\Delta\Theta_1(\gamma^1,\eta^1) = \eta^1 - \gamma^1 \qquad (4.2)$$

Para completar o mapa de configuração, com a segunda dimensão (Θ2) o elo 2 é rotacionado, determinando os espaços livres e proibidos, apenas para os espaços livres do elo 1 ($\Delta\Theta_i^l$, i = 1, ..., m). Esta segunda dimensão do mapa de configuração é expressa analiticamente na seguinte equação :

$$\Delta\Theta_2(\gamma^2,\eta^2,\Theta_1{}^l) = \Sigma^q_{r=1}\Delta\Theta_{2_r}{}^l(\gamma^l_{2_r},\eta^l_{2_r},\Theta_1{}^l) + \Sigma^s_{g=1}\Delta\Theta_{2^p_g}(\gamma^p_{2_g},\eta^p_{2_g},\Theta_1{}^l) \quad (4.3)$$

Combinando as duas equações obtém-se um mapa similar ao da figura 4.31, onde as áreas hachuradas representam os obstáculos ou áreas proibidas (de colisão) deixando o espaço restante para que o manipulador possa se deslocar livremente.

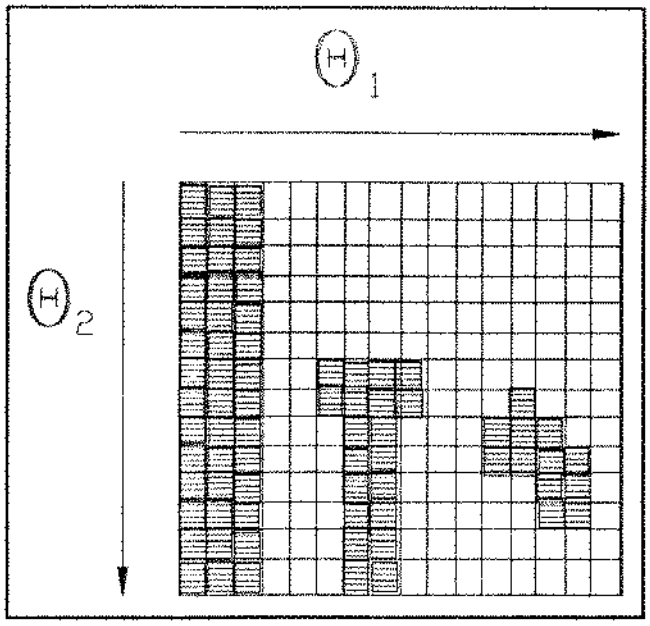


Fig. 4.31. Representação de um mapa de configuração

Em termos geométricos, o manipulador e os objetos existentes em uma dada configuração do espaço de trabalho são representados através de poliedros convexos. Para facilitar o processo de detecção de colisões, isto é feito considerando três planos horizontais (paralelos ao plano XY do sistema de referência do manipulador).

O primeiro plano de referência está determinado pela altura h1. Esta altura é medida desde o plano XY até a face inferior do poliedro que representa o atuador na posição DOWN (Figura 4.32).

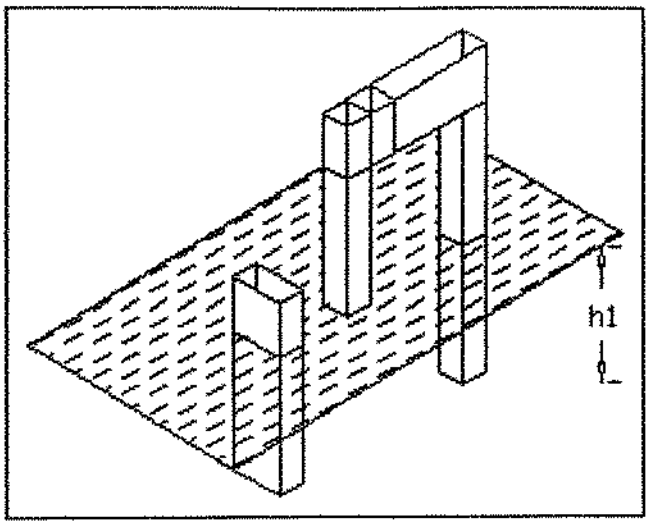


Fig. 4.32. Localização do plano de referência para h1

O segundo plano é definido de maneira similar, sendo que a altura que o determina é h2 (medida desde o plano XY até a face inferior do poliedro que representa o atuador na posição UP) (Figura 4.33).

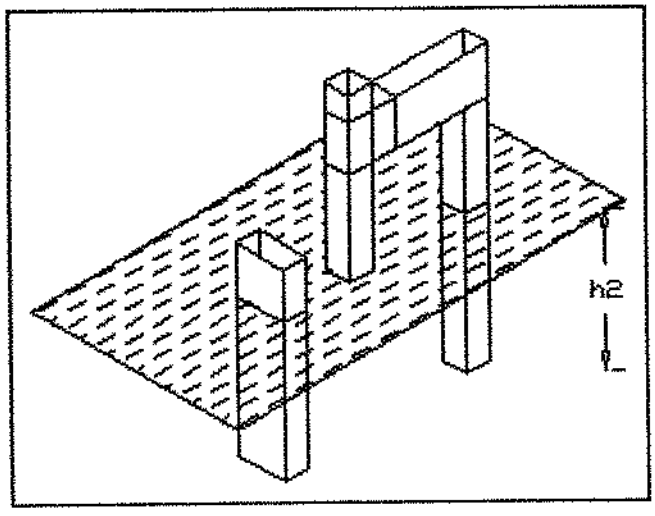


Fig. 4.33. Segundo plano de referência (h2)

O terceiro plano de referência é dado pela altura h3, definida pela perpendicular

entre o plano XY e as faces inferiores dos polihedros que representam os dois primeiros elos do manipulador (Fig. 4.34).

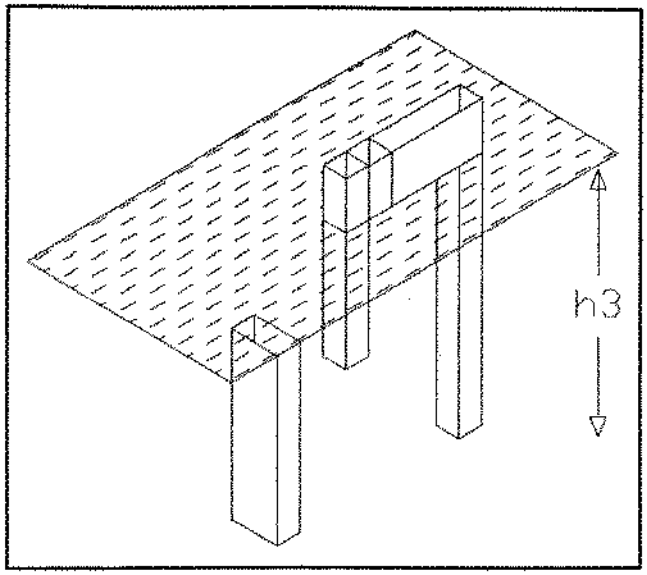


Fig. 4.34. Plano de referência para h3

Há a necessidade de se fazer teste de colisões para um dado objeto quando altura dele coincide ou ultrapassa algum dos planos de referência. Assim, se a altura do(s) objeto(s) ultrapassar h3 , o teste de colisões deverá ser feito nos três planos de referência. De igual forma, se a altura do objeto for maior que h2 e menor que h3, deve-se fazer o teste, apenas nos dois primeiros planos de referência. Entretanto, se algum objeto interseptar apenas o primeiro plano de referência, deverão ser testadas as possíveis colisões só nesse plano. Finalmente, se a altura de todos os objetos definidos for menor que h1, não existirão colisões para essa configuração do espaço de trabalho.

Em outras palavras, a idéia é fazer um teste de colisões no plano YZ e assim economizar tempo de cálculo, já que ao não existir colisões entre as projeções do manipulador e dos objetos neste plano, não será necessário fazer o teste nos planos paralelos ao plano XY. Esta técnica evita cálculos desnecessários incorporando critérios prévios para analizar a possibilidade de colisões dependendo das dimensões dos objetos.

Finalmente outro critério é incorporado para acelerar o processo de detecção. Este basea-se na localização dos objetos em relação à base do manipulador. Segundo isto, serão testados apenas os objetos que possuem um ou mais vértices dentro do raio determinado pelo comprimento dos dois elos do braço.

Para realizar o teste de colisões em qualquer dos três planos de referência, consideram-se áreas determinadas pela intersecção entre o manipulador e o plano, e pelos objetos com esse mesmo plano. Estas intersecções, se existirem, terão forma de polígonos convexos. As coordenadas dos vértices destes polígonos, são relativas a um sistema referêncial fixo à base do manipulador.

A detecção de colisão resume-se em determinar a existência de uma intersecção não nula entre os polígonos referentes aos obstáculos e aqueles referentes ao manipulador para uma dada localização deste Figura 4.35.

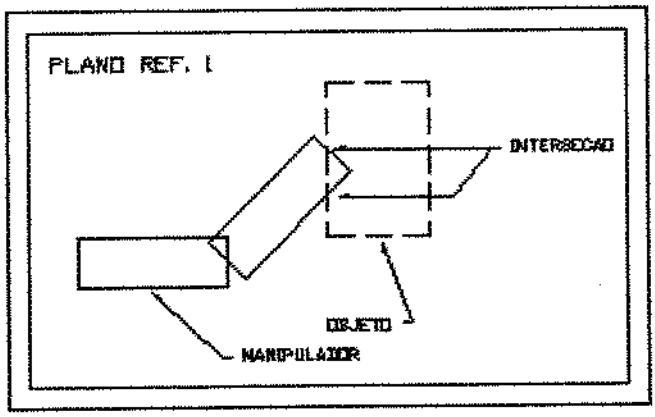


Fig. 4.35. Intersecção entre o manipulador e um objeto

Esta verificação é feita através de um teste de intersecção entre as arestas dos polígonos pertencentes ao manipulador e as arestas que determinam a projeção dos objetos no plano de referência, no qual está se desenvolvendo o teste. Resumindo, geometricamente, a detecção de colisões envolve o cálculo da interseção entre dois segmentos de reta.

Considerando dois segmentos de reta, e que estes estão definidos pelos seus pontos extremos, o cálculo da intersecção entre estes pode ser representado pelas seguintes equações **[RED-87]** :

Se o primeiro segmento, pertencente ao manipulador, for definido pelos pontos (x1,y1) e (x2,y2), a representação paramétrica deste segmento será :

$$x = x1 + c1\,(x2 - x1)$$
$$y = y1 + c1\,(y2 - y1) \qquad \text{com } 0 < c1 < 1 \qquad (4.4)$$

Da mesma forma, se o segundo segmento de reta, representando uma aresta do polígono resultante da intersecção de um objeto com o plano de referência, está definido pelos pontos (x3,y3) e (x4,y4), a intersecção entre estes dois segmentos de retas (arestas), pode ser deduzida segundo as seguintes equações:

$$x = x1 + c1\,\Delta X12$$
$$y = y1 + c1\,\Delta Y12 \qquad 0 < c1 < 1 \qquad (4.5)$$

$$x = x3 + c2\,\Delta X34$$
$$y = y3 + c2\,\Delta Y34 \qquad 0 < c2 < 1 \qquad (4.6)$$

Onde $\Delta Xij = Xj - Xi$

$\Delta Yij = Yj - Yi$

Igualando obtem-se :

$$X1 + C1\,\Delta X12 = X3 + C2\,\Delta X34$$
$$Y1 + C1\,\Delta Y12 = Y3 + C2\,\Delta Y34 \qquad (4.7)$$

Re-arranjando temos o seguinte sistema :

$$C1\,\Delta X12 - C2\,\Delta X34 = \Delta X13$$
$$C1\,\Delta Y12 - C2\,\Delta Y34 = \Delta X13 \qquad (4.8)$$

Resolvendo para C1 e C2 :

$$C1 = (\Delta XY34 + \Delta XY13 + \Delta XY41) / D$$
$$C2 = (\Delta XY32 + \Delta XY13 + \Delta XY21) / D \qquad (4.9)$$

Com $\Delta XYij = XiYj - XjYi$

$D = \Delta X21\Delta Y34 - \Delta X34\Delta Y21$

Portanto, se D <> 0, 0 <= C1 <= 1, 0 <= C2 <= 1, existe uma intersecção cujas coordenadas são determinadas substituindo-se o valor de C1 nas equações (4.5) ou de C2 nas equações (4.6).

Um aspecto importante na construção do mapa de configuração é a resolução deste. É praticamente impossível representar explicitamente o espaço de configuração de uma forma realística. O realismo será frequentemente limitado pela capacidade de memória e a disponibilidade de tempo computacional para o cálculo. Noborio e Watanabe **[NOBO-89]** afirmam que ao decompor o espaço total de uma junta rotacional em apenas 64 intervalos, serão necessários $6.9 * 10^9$ testes de interferências entre o manipulador e os objetos existentes no envelope de trabalho. Mesmo assim, serão necessárias $6.9 * 10^9$ bytes na memória do computador para representar este espaço de configuração. Pode-se apreciar facilmente que o processo de construção de um mapa de configuração é um procedimento demorado e altamente consumidor de memória.

Como o mapa de configuração é uma sucessão de posições do robô em relação a uma diposição de objetos, caso a resolução definida pelo usuário for grosseira ("quanta" de percurso angular muito grande) corre-se o risco de eventualmente "saltar" uma colisão intermediária entre duas posições.

O sistema dá a possibilidade ao usuário definir a resolução do mapa de configuração, dependendo do grau de segurança que ele precisar ou das capacidades de memória e disponibilidade de tempo computacional para o cálculo. Assim, o usuário pode dividir o intervalo angular dos dois primeiros elos, nos intervalos que ele desejar.

5. CONCLUSÕES E PERSPECTIVAS :

O sistema apresentado cumpre os objetivos inicialmente planejados de prover uma ferramenta de apoio à programação "off-line" de robôs industriais. Através do uso desta ferramenta computacional as trajetórias do manipulador poderão ser definidas e testadas considerando as restrições no envelope de trabalho.

Desta forma, o sistema ajudará no "try-out" de programas, permitindo otimizar esta tarefa e diminuindo o tempo de preparação de novas aplicações. Este fator torna-se o principal atrativo para a sua implantação no ambiente industrial.

Do ponto de vista acadêmico, tem-se uma ferramenta de apoio ao ensino da robótica. Assim, os estudantes terão contato direto com ferramentas de manufatura assistida por computador, as quais poderão, em algum caso, suprir a existência física do próprio manipulador. Isto permitiria o acesso dessas técnicas a laboratórios que atualmente não possuem recursos adequados para este propósito.

Apesar de outros pacotes similares existirem no mercado, de fato alguns deles mais sofisticados, o sistema desenvolvido prova ser efetivo tanto em termos econômicos quanto pela simplicidade operacional.

Outra das vantagens do sistema desenvolvido é a acessibilidade ao código fonte principalmente por ter sido implantado num "hardware" simples e de amplo uso (PC compatíveis), usando técnicas de programação estruturadas e alto nível combinado com procedimentos gráficos pré-definidos e altamente difundidos (TURBO GRAFIX TOOLBOX). Isto permite um fácil aprimoramento e modificação para se encaixar às necessidades de usuários específicos.

Futuras melhorias estão previstas para permitir a incorporação de outras formas geométricas na representação dos objetos presentes no envelope de trabalho. Pretende-se assim, incorporar pirâmides e cones, obtendo uma melhor visualização na simulação das aplicações e uma detecção mais exata das possíveis colisões durante a execução das mesmas.

Com respecto à interpretação dos comandos da linguagem de programação no futuro deseja-se testar técnicas de inteligência artificial que permitam otimizar este processo. De

fato, a técnica de representação de gramáticas de livre contexto parece ser um bom ponto de partida para futuros desenvolvimentos de pré-processadores mais eficientes e rápidos.

No futuro serão escritos pré-processadores para outras linguagens compatíveis como o robô em uso. Isto permitirá uma validação mais geral do conceito de interface neutra para a comunicação entre o módulo pré-processador e os módulos de representação gráfica.

APÊNDICE

A linguagem de programacao AML :-

AML é uma linguagem de programação estruturada e interativa de alto nível, projetada pela IBM. Esta fornece funções típicas dentro das linguagens de propósito múltiplo, oferecendo uma alta variedade de tipos de dados e operadores, além de estruturas de controle e, comandos de entradas e saídas (I/O) possibilitando assim, uma fácil integração de sistemas. Adicionalmente AML fornece recursos básicos de sensoreamento e comunicação, que são de vital importância para completar as tarefas pelo manipulador.

Estas funções básicas permitem ao programador construir uma estrutura sobre os elementos de controle do robô para maximizar as capacidades de controle do programa e minimizar os esforços de programação.

Ainda mais, AML possui um conjunto de sub-rotinas para o gerenciamento de arquivos. Estes elementos permitem ao programador gerar sub-rotinas próprias e usá-las conjuntamente ou no lugar daquelas fornecidas pela AML. Estas subrotinas podem ser agrupadas ou usadas recursivamente, conseguindo criar um ambiente "customizado", no qual pode-se escrever programas de aplicação ótimos e de maneira muito simples.

Os comandos em AML são semelhantes a uma frase em inglês, e podem ser classificados em quatro grupos segundo o seu tipo :

- ○ Comandos de Declaração de Variáveis.
- ○ Comandos Executáveis.
- ○ Comandos Sensoriais.
- ○ Outros comandos.

1.Comandos de declaração de variáveis :

AML possui 4 tipos de variáveis específicas às aplicações : "Values", contadores, pontos e "aggregates". Estas variáveis sempre devem ser declaradas no começo do programa ou da subrotina onde serão usadas. A forma de declará-las é similar para os quatro tipos.

Os componentes do comando de declaração de variável são os seguintes : o nome da variável (chamado 'id', na linguagem AML) ou NEW ou STATIC que são as duas possibilidades de comando que irão identificar que essa instrução é uma declaração, e portanto o interpretador deverá avaliar tal comando para determinar o tipo de variável que está sendo declarada e o seu valor inicial. Um ponto e vírgula indica o final do comando. Os exemplos abaixo fornecidos mostram diferentes tipos de comandos de declaração de variáveis :

```
X0: NEW 56;
X1: NEW 12;
X2: NEW PT (123.0,45.0,0);
X3: STATIC COUNTER;
X4: NEW < X0 , X1 >;
```

Os dois primeiros exemplos declaram variáveis do tipo "value"; a variável X2 será identificada pelo compilador como uma coordenada cartesiana onde o manipulador irá desenvolver alguma tarefa; X3 será um contador e finalmente X4 será considerado como um "Aggregate" que agrupa num conjunto os dois "values" declarados previamente.

A seguir serão dadas explicações mais detalhadas de cada um dos tipos de variáveis que o AML pode usar, com as restrições que o compilador impõe para cada uma delas, e as situações mais comuns onde estas são usadas.

Os "values" são números inteiros ou reais aos quais é designado um nome para o posterior uso desse valor de uma maneira simbólica, facilitando assim a identificação por parte do programador e uma posterior depuração dos programas. A definição de um "value" tem a seguinte forma :

A :NEW 5;

VALOR1 :NEW 5.8;

Os nomes dos "values" devem seguir as seguintes restrições: comprimento máximo de 72 caracteres, o primeiro caracter deve ser sempre alfabético, os caracteres seguintes podem ser letras, números o sublinhado (_).

Os contadores, a diferência dos " values", podem ter os seus valores mudados através da operação do programa, permitindo que o controlador mantenha uma descrição interna dos processos externos atuais. Por exemplo, é possível produzir um determinado número de peças, ou até um certo nível de pecas defeituosas; como estas entidades são estáticas e o controlador apenas retém o último valor ou valor atual de cada contador, inclusive caso ocorra uma queda na energia elétrica, é possível medir quantidades produzidas, diariamente ou mensalmente, por exemplo.

A faixa permitida para os valores dos contadores está entre -32767 e +32767, admitindo apenas números inteiros.

O termo STATIC indica que o nome da variável deve ser atribuido a um contador. Também implica que a variável não tem um valor inicial específico, e que o usuário deverá setar esse valor inicial através de outra instrução (SETC).

As regras para nomear os contadores devem ser as mesmas que são usadas para os values.

Os pontos são variáveis que contém a informação necessária para os comandos de movimentos (vide Comandos executáveis). Principalmente essa informação está referida as coordenadas cartesianas de uma localização no espaço de trabalho e o ângulo de rotação da garra para essa localização. Uma vez que um ponto é declarado o controlador não verificará se esse ponto é ou não atingível pelo manipulador; esse teste só será feito no momento da execução do programa.

Para declarar um ponto, o comando NEW deve ser seguido do termo PT (indicando ao compilador que a entidade que está sendo declarada é um ponto), a seguir e entre parênteses devem estar indicadas as coordenadas XY do ponto que deseja-se definir e a orientação da garra (ângulo roll). Um exemplo de declaração de ponto é mostrado abaixo :

```
PONTO_1 : NEW PT (125.0,45.8, 0);
PONTO_2 : NEW PT (650,0,0);
```

As regras para definir os nomes dos pontos são as mesmas que as usadas na definição de "values" e contadores.

Finalmente existe uma forma de agrupar em um conjunto diversos elementos (de um mesmo tipo) e assim poder usá-los de maneira integrada, principalmente com comandos de tipo iterativo. Este tipo de variável facilita enormemente a tarefa de programação de 'loops' e ações repetitivas.

A definição de um "aggregate" está condicionada à existência de todos os elementos nele contidos, ou seja todos os elementos que serão embutidos num conjunto devem estar já definidos e obviamente devem ser do mesmo tipo. A seguir, um exemplo completo de uma definição de um "aggregate" de pontos e outro de values :

```
V1 : NEW 4;
V2 : NEW 6;
V3 : NEW 8;
P1 :NEW PT (650,0,0);
P2 : NEW PT (0,650,180);
AGGR1: NEW < V1, V2 ,V3 >;
AGGR2: NEW <P1 , P2 >;
```

No exemplo acima são declarados dois conjuntos, o primeiro AGGR1 contém tres valores (V1,V2,V3) e o segundo "aggregate", AGGR2 contém dois pontos (P1,P2). Os nomes dos "aggregates" devem seguir as mesmas regras que os tipos de variáveis anteriores.

Em alguns comandos do AML, especificamente nos que alteram o fluxo do programa (desvios condicionais ou incondicionais), são referênciados blocos ou linhas às quais é transferido o controle de fluxo. Isto é equivalente ao endereço de um bloco de programa,

similar a outras linguagens de propósito múltiplo (por exemplo o GOTO em BASIC). Este endereço é chamado de LABEL. Os LABELs devem ser especificados, seguindo as mesmas regras de designação dos tipos de variáveis anteriores e são identificados pelo compilador através de dois pontos (:) colocados a seguir do nome do LABEL. A seguir são apresentados exemplos do uso de LABELs num programa em AML :

```
SAI : END;
LABEL1 : DELAY (12.9);
```

A existência de um comando em seguida de uma etiqueta ou LABEL, não é obrigatória, podendo ocorrer na seguinte forma:

```
SAI :
END;
```

2. Comandos Executáveis :

A linguagem AML foi projetada especificamente para a programação de manipuladores; por essa razão é que ela possui vários comandos destinados à movimentação do braco por completo ou de algum dos seus componentes independentemente. Além desses comandos existem alguns outros que tem uma estreita relação com os comandos de movimentos alterando algumas das suas características ou adicionando algum parâmetro necessário para a realização adequada das ações. Devido a isto é que serão incluidos nesta seção.

Existem dois modos de movimentar o braço do manipulador de acordo ao tipo de movimento: Absoluto e/ou incremental.

No movimento absoluto o programador deve identificar o ponto meta desejado em coordenadas cartesianas absolutas, ou seja de acordo ao referêncial localizado na base do braço (referencial zero). Um exemplo é dado a seguir :

PMOVE(PT (0,650,0));

Este comando levará ao braço até a posição x = 0, y = 650 com um ângulo roll da garra de zero grau.

Em movimento incremental o programador deve identificar o movimento em função dos intervalos (em mm) que o braço deverá percorrer entre o ponto atual e o ponto meta desejado. Por exemplo se o manipulador está na posição X = 100 , Y = 450 com um ângulo roll de 180º e a posição meta desejada é X = 0 , Y = 650 com um ângulo roll de 0º , o comando necessário será o seguinte :

DPMOVE (< -100, 200, -180>);

Ambos os comandos podem usar variáveis para identificar os seus parâmetros, por exemplo :

PMOVE (P1);
DPMOVE (< V1 , V2 , V3 >);

Assim se um ponto vai ser usado várias vezes dentro de um programa, o programador evitará escreve-lo de maneira explícita todas as vezes.

Associado a um movimento existem conceitos importantes que alterarão o tipo de trajetória, a velocidade e a precisão de posicionamento final.

O primeiro destes conceitos é o PAYLOAD que associa o conceito de carga que o manipulador irá deslocar com a velocidade à qual ele vai realizar o movimento.

Em outras palavras o programador pode especificar através do uso deste comando a velocidade dos movimentos que o robô irá realizar.

A velocidade é função do valor especificado entre parênteses no comando PAYLOAD, do tipo de manipulador e do peso da carga que será manipulada. O valor pode ser especificado diretamente através de um número inteiro, com valores entre 0 e 10 ou através de

uma variável, a seguir são mostrados dois exemplos desse comando :

PAYLOAD (10) Máxima velocidade
PAYLOAD (1) Mínima velocidade
PAYLOAD (V1) Uso de uma variável

Para efeitos da simulação estes valores não serão considerados, e os movimentos do robô serão mostrados sempre na mesma velocidade.

Outro comando que influi principalmente sobre o tipo de trajetória e indiretamente sobre a velocidade dos movimentos é o comando LINEAR.

O comando LINEAR permite ao programador definir trajetórias lineares, ou seja, o braço seguirá um percurso retilíneo entre o ponto origem e o ponto destino. Quando este comando é usado, todas as trajetórias serão retilíneas , além disso, as velocidades definidas pelo usuário serão redefinidas com valores apropriados escolhidos pelo próprio controlador dependendo da precisão da interpolação linear seleccionada pelo programador. Esta precisão ou qualidade do movimento retilíneo é representada por um número inteiro entre 1 e 50. O valor da qualidade representa o desvio ou erro em função de uma trajetória reta, então um baixo valor de qualidade representa uma trajetória muito próxima à linha reta, porém isto derivará em movimentos mais lentos. Inversamente, quando o valor de qualidade aumenta, o movimento tende a se afastar do movimento retilíneo, obtendo-se assim uma velocidade maior. Ao desativar o comando LINEAR (definindo uma qualidade 0, valor default) os movimentos do manipulador serão interpolados no espaço das juntas. Este tipo de movimentos terão trajetórias imprevistas dependendo dos pontos origem e destino.

O valor da qualidade do comando LINEAR pode ser definido por um número inteiro diretamente ou com o uso de uma variável. A seguir são mostrados exemplos deste comando :

LINEAR (0); Movimento no espaço das juntas
LINEAR (1); Trajetória retilínea precisa
LINEAR (50); Trajetória mais rápida, porém com o maior erro.

LINEAR (V1); Uso de variável.

Finalmente outro comando que influi sobre os movimentos do manipulador é o comando ZONE. Através deste, o programador pode especificar a precisão de posicionamento do manipulador ao atingir um ponto dado. O valor da precisão é representado por um número entre 0 e 15. Onde o valor menor indica a maior precisão possível, requerendo, portanto um tempo maior para completar o movimento.

O fator de precisão do comando ZONE pode ser diretamente definido através de um número inteiro ou por meio de uma variável com valor que esteja no intervalo permitido pelo comando. A seguir são apresentados dois exemplos para o uso deste comando :

ZONE (1); Preciso, porém lento
ZONE (15); Impreciso, portanto rápido
ZONE (V2); Uso de uma variável.

O movimento perpendicular ao plnano XY (mesa) do robô IBM 7535 possue apenas duas posições, alta e baixa. Dai inclusive, a denominação de meio eixo para este. Modelos mais avançados do mesmo fabricante possuem um servo motor que comanda o posicionamento deste terceiro elo, de uma forma contínua. Considerando, no nosso caso o IBM 7535, estas duas posições podem ser comandadas através da linguagem AML com o uso dos comandos UP e DOWN. O comando UP levantará o elo e o atuador segundo o eixo Z até a posição superior. Igualmente o comando DOWN fará o elo e o atuador descer até a posição mais baixa permitida. Estes dois comandos podem estar acompanhados de um parâmetro que indicará o tempo máximo de espera para a confirmação de que a operação foi completamente realizada. O tempo 'default' é de 1.5 segundos, porém o programador pode usar valores entre 0 (espera infinitamente) até 25.5 segundos. Assim se a operação não for completada com sucesso no intervalo de tempo especificado, será gerada uma condição de erro no controlador do robô.
O tempo de espera pode ser especificado através de um número diretamente o por meio de uma variável. A seguir são mostrados exemplo destes comandos :

DOWN;Usa o valor default.

DOWN (23.2);Espera pela confirmação por 23.2 segundos.

UP;

UP (V3); Uso de variáveis.

Para comandar os movimentos de abre/fecha da garra o AML possue dois comandos que realizam estas duas operações diretamente : GRASP que fecha a garra e RELEASE que a abre.

Outro comando usualmente relacionado aos comandos de movimento é o DELAY. Este comando suspende a execução do programa durante um tempo especificado, entre parénteses como parâmetro. Estas suspenções são frequentemente necessárias junto à execução dos comandos GRASP e RELEASE, permitindo ao programador ter maior certeza de que os movimentos da garra sejam completamente executados antes de continuar com a execução dos comandos subsequentes. O valor da espera deve estar na faixa de 0 a 25.5 segundos. Ao igual que os comandos anteriores o parâmetro pode ser definido diretamente como un número ou através de uma variável pré-definida. Veja a seguir exemplos do uso de este comando :

```
DELAY ( 0 );
DELAY ( V3 );
```

3.Comandos Sensoriais :

Outra das características principais do AML é a possibilidade de manusear e interpretar informações referente ao ambiente no qual a aplicação está se desenvolvendo. É assim que a linguagem possue três comandos específicos para trabalhar com as informações vindas de sensores para gerar informações de saídas em atuadores ligados à aplicação em execução.

O primeiro desses comandos é o TESTI, o qual testa numa porta de entrada dada (DI), a existência de um valor lógico específico. Se o valor achado na porta coincide com o especificado pelo programador como parâmetro no comando TESTI, o controle do programa é

transferido para o início de um bloco do programa especificado através de uma etiqueta (label). Se o valor do parâmetro for 0 a transferência do controle do programa só ocorrerá se o controlador achar uma condição de aberto na DI especificada. Por outro lado, se o valor for diferente de zero, a transferência acontecerá só se o controlador achar uma condição fechada na porta de entrada dada.Este teste é instantâneo, ou seja o controlador não espera para que a condição seja verdadeira; então, se a condição detectada for falsa, a execução do programa continuará na linha seguinte.

A sintaxe do comando TESTI é a seguinte:

TESTI (DI , VALUE , LABEL);

Onde DI indica o número da porta de entrada lógica, sob a qual será feita a comparação. Esta porta pode ser identificada por um número diretamente ou por meio de uma variável inteira. VALUE indica o valor necessário para que o desvio condicional ocorra se este é encontrado na porta de entrada especificada (Da mesma forma que o DI, este valor pode ser especificado tanto por meio de um número ou pelo uso de uma variável). O LABEL indica a direção à qual o controle do programa será transferido se a condição é cumprida.

O segundo comando do tipo sensorial é o WAITI, este faz o controlador esperar por um valor na entrada lógica específica. Esta espera se extende até 25.5 segundos, em incrementos de 0.1 segundos. Se o tempo de espera especificado for 0 o controlador esperará indefinidamente pela entrada lógica definida como parâmetro no comando. Ao expirar o tempo de espera o programa detém e se produz uma condição de erro no console do controle do manipulador. A sintaxe deste comando é a seguinte :

WAITI (DI , VALUE , TIME-LIMIT);

Os três parâmetros desse comando são especificados tanto por meio de um número ou indiretamente através do uso de variáveis.

O último comando sensorial gera uma saída lógica para um atuador ligado ao controlador do sistema. Assim, um valor 0 abrirá o relé associado à saída especificada pelo DO,

como um valor diferente de zero ativará o fechamento do relé ou atuador ligado ao DO especificado. A sintaxe desse comando é a seguinte :

WAITI (DO , VALUE);

Ambos os parâmetros podem ser especificados tanto por números inteiros como também por meio de variáveis.

Finalmente no próximo ponto são apresentados outros comandos de difícil classificação, que estão relacionados com o controle do fluxo e a depuração dos programas fazendo-os mais eficientes e fáceis de entender.

4.Outros comandos :

Temos subdividido essa categoria de comandos sem classificação em três tipos. O primeiro tipo tem relação com a transferência de controle de fluxo dentro de um programa. Isto é, existem comandos que permitem ao usuário programar desvios na sequência de instruções para um bloco o linha de programa localizado em outro local do programa.

O primeiro desses comandos é o BRANCH, que permite especificar um desvio incondicional para uma linha de programa identificada através de um rótulo ou etiqueta. A única limitação desse desvio é que a linha ou bloco destino deve estar declarado no mesmo nível de subrotina o qual contem o BRANCH, porém ela pode estar declarada antes ou depois da linha que ativa o desvio. A sintaxe do BRANCH é simples e direta, como pode-se ver no seguinte exemplo :

```
BRANCH ( ETIQ1 );
DELAY ( 20 );
PMOVE ( HOME );
ETIQ1 : PMOVE ( LOCAL1 );
END;
```

Nesse exemplo a execução da primeira linha transferirá o controle à linha etiquetada como ETIQ1, produzindo um salto incondicional das duas linhas seguintes. Assim a instrução que será executada em seguida do BRANCH será o PMOVE (LOCAL1); continuando com a execução sequencial a partir dessa linha até o final do bloco ou até achar outro comando de desvio.

Outro comando que altera a sequência natural de execução de um programa, produzindo um desvio é uma chamada a subrotina.

Uma subrotina é um subprograma que realiza uma tarefa específica. Este subprograma está composto por um grupo de sentências às que asigna-se um nome e constitui uma unidade em si. A tarefa atribuida à subrotina pode ser executada diversas vezes, através da simples chamada do seu nome.

As subrotinas podem tomar uma das seguintes formas:

```
id : SUBR;
 comando 1
 comando 2
 :
 :
 comando n
END;
```

```
id : SUBR ( param1, param2, ... paramN );
 comando 1
 comando 2
 :
 :
 comando N
 END;
```

Como pode ser apreciado no segundo exemplo o AML permite a transferência de parâmetros ao interior de uma subrotina, permitindo o uso dela para diversas situações ou ocorrências de um parâmetro. A chamada de uma subrotina com parâmetros é feita da seguinte forma:

```
NOMESUBR1( PARAM1, PARAM2, ...., PARAMN );
```

É necessario salientar que o tipo de cada parâmetro de chamada deve coincidir com o tipo e o uso que terá cada um deles ao interior da subrotina. Os tipos de dados que o AML permite transferir como parâmetros ao interior de uma subrotina são : Pontos, Contadores e Values.

O segundo tipo de comandos especiais está destinado à definição e manipulação de contadores. Esses contadores permitem ao programa manter uma descrição quantitativa interna do estado do processo externo. Por exemplo, a execução de uma determinada tarefa por um número fixo de vezes, ou a contagem do número de peças defeituosas num processo específico. Existem quatro comandos que estão relacionados ao uso de contadores. O primeiro permite definir o valor para um contador determinado. Esta definição pode ser usada quantas vezes o programador precisar dela. A sintaxe deste comando é a seguinte:

```
SETC ( CONT1 , 1 );
```

Onde CONT1 representa o nome do contador que está sendo "setado" no valor "1". O valor pode estar representado como um número diretamente ou através de uma variável pré-definida. É necessário destacar que para usar este comando o contador já deve ter sido declarado pela palavra STATIC COUNTER (Veja seção 1 desse apêndice).

O valor de um contador pode ser incrementado ou decrementado segundo os requerimentos da aplicação. Isto é feito pelos comandos INCR e DECR respectivamente. A sintaxe de ambos os comandos é igual, como podemos ver a seguir :

```
INCR ( CONT1 );
```

DECR (CONT2);

No exemplo o valor do contador CONT1 será incrementado em uma unidade e o valor do CONT2 diminuirá em um o seu valor.

Finalmente o programador pode definir desvios condicionais dependendo do valor atual de um contador. Isto é feito através do comando TESTC que compara o valor do contador referênciado com o valor dado, ativando uma transferência no fluxo do programa se ambos valores coincidirem. O endereço da transferência do controle está definido pelo LABEL especificado como parâmetro no comando. Veja a seguir a sintaxe deste comando :

TESTC (NOME , VALOR , LABEL);

NOME indica qual o contador cujo valor será comparado com VALOR. Este segundo parâmetro pode estar definido como um número diretamente ou através do uso de uma variável. Finalmente LABEL indica a nova direção do fluxo se a comparação fosse verdadeira.

O terceiro grupo de comandos é chamado de iterativos e permitem ao programador repetir uma chamada a subrotina ou um outro comando simples, usando como parâmetro um ou vários conjuntos de valores agrupados em um aggregate (Veja seção 1 desse apendice). Ao executar um ITERATE só é possível abandonar o comando, quando a totalidade dos elementos do ou dos aggregates tem sido usada. O formato do ITETRATE pode ter uma das seguintes formas :

ITERATE ('comando' , < aggregate1 >, ... , < aggregateN >);
ITERATE ('subrotina', < aggregate1 >, ... , < aggregateN >);

É importante destacar que se o programador está usando o comando ITERATE com mais de um aggregate como parâmetro, ele deve ter a certeza que todos os aggregates possuem a mesma quantidade de elementos, e que esses elementos estão alinhados nos aggregate na ordem apropiada à aplicação.O manual de programação do AML\E não especifica quais os comandos que podem ser usados com o ITERATE. Considerando isto, foram desenvolvidas

provas com o compilador para determinar o conjunto de comandos que realmente fornecem certo nível de utilidade ao serem usados na forma iterativa. A seguir são apresentados os oito casos nos quais o compilador do AML não detetou erro de sintaxe e, por tanto, poderiam ser usados sem maior problema pelo programador :

ITERATE ('SETC', param1 , param2); Onde
Param1 pode ser um contador ou um aggregate de contadores.
Param2 pode ser um número, uma variável pré-definida, um
aggregate de números ou um aggregate de variáveis pré definidas.

ITERATE ('INCR', param1);
ITERATE ('DECR', param2); Onde

Param1 só pode ser um aggregate de Contadores.

ITERATE ('TESTC', param1, param2, param3);Onde

Param1 sp pode ser um aggregate de contadores.
Param2 pode ser um número, uma variável pré-definida, um
aggregate de números ou um aggregate de variáveis
pré-definidas.
Param3 só pode ser uma LABEL.

ITERATE ('TESTI', param1, param2, param3);Onde

Param1 pode ser um número, uma variável
pré-definida, um aggregate de números ou um
aggregate de variáveis pré-definidas.
Param2 pode ser um número, uma variável
pré-definida, um aggregate de números ou um

aggregate de variáveis pré-definidas.

Param3 só pode ser uma LABEL.

ITERATE ('WAITI', param1 , param2, param3); Onde

Param1 pode ser um número, uma variável pré-definida, um aggregate de números ou um aggregate de variáveis pré-definidas.

Param2 pode ser um número, uma variável pré-definida, um aggregate de números ou um aggregate de variáveis pré-definidas.

Param3 só pode ser uma LABEL.

REFERÊNCIAS BIBLIOGRÁFICAS

[BEJC-85] Bejczy,A.K. "Control of remote manipulators" em Handbook of Industrial Robotics" Nof, S.Y.(Ed.) pp. 320-336 USA 1985

[BLUM-90] Blumenthal,T. "Economic effects of robotization in Japan.". Robotics and Autonomus Systems. V6 n4 pp. 323-326. 1990.

[BONN-87] Bonney, M.C., R.J.Marshall e J.L.Green. "Off-line programming using the GRASP robot simulation system" em "Off-line programming of Industrial Robots". Storr,A. e J.F.McWaters (Ed.). pp. 171-179. 1987

[BRAN-91] Brandão,M.A.L. e P. Seleghim. "Robótica na Coréia do Sul". Revista de Administração. V26 n3 pp. 73-77. 1991

[BUCK-85] Buckley, S.J. e G.F.Collins. "A structured pogramming robot language" em "Handbook of Industrial Robotics" Nof, S.Y.(Ed.) pp. 381-403. USA 1985

[CHAN-88] Chan, S.F., R.H.Weston e K.Case. "Robot simulation and off-line programming".Computer-Aided Engineering Journal. V5 n4pp.157-162. 1988

[COUR-81] Courtney,J. e G.Herlin. "A portable Robot Software System". Proceedings 11th International Symposium on Industrial Robots. pp 577-584. Japan 1981.

[CRAI-85] Craig, J.J. "Introduction to robotics : Mechanics and Control." Addison-Wesley Publishing Co.Inc. 2da.Edição. 1989

[DEIS-85] Deiseroth,M.P. "Robot teaching" em "Handbook of Industrial Robotics" Nof,

S.Y.(Ed.) pp. 352-365 USA 1985

[DENA-55] Denavit,J. e R.S.Hartemberg. "A kinematic notation for lower-pair mechanis based on matrices." Jounal of Applied Mechanics. pp. 215-221. Jun 1955.

[EDKI-85] Edkins,M. e C.R.T.ith."The practical problems involved in off-line programming a robot from a CAD system." em "Robots and Automated Manufacture." Billingsley, J. (Ed.).pp. 29-39. U.K. 1985

[ERTH-91] Erthal, J.L. e N.Back. "Representação gráfica de robôs industriais de cadeia aberta". Annais do XI congresso Brasileiro de Engenharia Mecanica. pp. 525-528. Brasil 1991.

[EUSE-84] Eusebio,M.J.C. e J.L.Braga. "NATFOR : Um pré-processador para o FORTRAN.". Annais do IV Congresso da Sociedade Brasileira de Computação.pp. 289-299.Brasil 1984.

[FELD-91] Feldmann,K. "Computer-Aided Planning and Control of automated assembly systems". Siemens Review n3 pp. 16-20. 1991

[GOLD-84] Goldesão.A. e C.L.Candiani "Um compilador para control numérico". Annais do I Congresso Latino-Americano de Automática. pp. 1214-1219. 1984

[GROO-89] Groover,M.P., M.Weiss, R.N.Nagel e N.G.Odrey. "Robótica - Tecnologia e Programação".McGraw- Hill 1988.

[GRUV-84] Gruver, W.A., B.I.Soroka, J.Craig e T.L.Turner. "Industrial Robot Programming Languages : A comparative evaluation" IEEE Transactions on Systems, Man and Cybernetics. V C-14 n4 pp. 565-570. 1984

[HALM-88] Halme, A., A.Visala e T.Torvikoski. "An interactive task oriented robot

programming system." em "IFAC software for computer control" Macleod, I.M. e A.D.Heher (Ed.).pp. 127-133. S.Africa 1988

[HEUB-90] Heuberger,C.F. "Determinação Automática de Trajetórias ótimas para um manipulador na prescença de obstáculos." Tese de Mestrado UNICAMP 1990.

[IBM-83] IBM-7535/7540 Manufacturing System Hardware Library 2da.Edição. USA 1983.

[KARA-91] Karami,M. "3-D Simulation : A new industry standart". Industrial Engineering pp. 36-38 Jul. 1991

[KERN-76] Kernigam, P. "Sotware Tools"

[KOPA-88] Kopacek,P. e I.Troch. "Software for Simulation and Control of Robots and Manipulators." IFAC Software for Computer Control. pp. 21-24. S.Africa 1988.

[LANG-87] Langrudi,S.A. "VALSIM.LU : A graphic simulator of the off-line programming language VAL-II". em "Off-line programming of Industrial Robots". Storr,A. e J.F.McWaters (Ed.). pp. 119-134. 1987

[LEE-90] Lee,D.M.A. e W.H.ElMaraghy. "ROBOSIM : A CAD-based off-line programming and analysis system for robotic manipulators." Computer-Aided Engineering Journal. V7 n5 pp.141-147 1990

[LOZA-83] Lozano-Pérez, T. "Spatial planning : A configuration space approach." IEEE transactions on Computers. V c-32 n2 pp. 108-119. 1983

[MAYE-87] Mayer,R.J."IGES: One answer to the problems of CAD database exchange". Byte Jun.1987 pp. 209-214.

[MEIJ-88] Meijer,G.R. e L.O.Hertzberger. "Off-line programming of exception handling estrategies". IFAC Robot Control pp.431-436. FRG 1988.

[MEND-90] Mendeleck, A. e D.E.Zampieri. "Desenvolvimento de um controle supervisor para um manipulador antropomorfico com 6 graus de liberdade." pp.170-175. 1990

[MILB-88] Milberg,J., N.Schrtfer e A.Tauber. "Requirements for Advanced Graphic Robot Programming Systems". IFAC Robot Control. pp. 487-492. FRG 1988.

[MILB-89] Milberg,J. e H.Diess. "Assembly Simulation - an efficient tool for assembly-oriented design" Annals oà the CIRP. V38 n1 pp. 21-24.1989

[MUNS-85] Munson, G.E. "Industrial Robots : Reliability, maintenance and safety" em "Handbook of Industrial Robotics" Nof, S.Y.(Ed.) pp. 722-758 USA 1985

[NOBO-89] Noborio H. e W.Watanabe. "A fast Motion-planning algorithm running on a all graph with collision-free ranges in the joint space." Proceedings oh the 28th SICE Annual conference. V2 pp. 1305-1308. Japan 1989.

[NWAG-91] Nwagboso, C.O. e J.C.Whitehouse. "Prediction of robot cycle time of threaded components-quality condition monitoring". Proceedings of the Institution oà Mechanical Enginners.V205 pp. 51-57. 1991

[OTA-84] "Computerized Mannufacturing Automation : Employment, Education and the Workplace". U.S.Congress, Office of Technology Assesment OTA-CIT-235 Washington D.C. USA 1984.

[POLL-87] Pollman, W.e H.Dzembritzki. "Off-line programming of Industrial Robots" em "Off-line programming of Industrial Robots". Storr,A. e J.F.McWaters (Ed.). pp. 7-18. 1987

[RED-85] Red, W.E. e H.V.Truong-Cao. "Configuration Maps for Robots Path planning in Two Dimensions." Transactions of the AE. v107 pp. 292-298. 1985

[RED-87] Red, W.E., H.V.Truong-Cao e K.H.Kim. "Robot path planning in Three-Dimensions using the direct subspace" Transactions of the AE v109 pp. 238-244. 1987

[REMB-87] Rembold,U. "Programming of Industrial Robots. Today and in the future" em "Languages for sensors-based control in robotics" Rembold,u. e K.Hormann (Ed.) pp. 3-23. FRG 1987

[SALV-85] Salvendy, G. "Human Factors in Planning Robotic Systems" em "Handbook oà Industrial Robotics" Nof, S.Y.(Ed.) pp. 639-664 USA 1985

[SMIT-85] Smith, R.C. e D.Nitzam." Modular programmable assembly research." em "Handbook of Industrial Robotics" Nof, S.Y.(Ed.) pp. 1151-1172. USA 1985

[STOB-85] Stobart, R.K. e C.Dailly. "The use of simulation in the off-line programming oà robots" em "Robots and Automated Manufacture." Billingsley, J. (Ed.).pp. 29-39. U.K. 1985

[STOB-87] Stobart,R.K. "Collision detection for off-line programming of robots." em "Off-line programming of Industrial Robots". Storr,A. e J.F.McWaters (Ed.). pp. 107-117. 1987

[STOR-87] Storr, A. e H.Schumacher. "Programming methods for Industrial Robots" em "Off-line programming of Industrial Robots". Storr,A. e J.F.McWaters (Ed.). pp. 1-4. 1987

[SZPA-87] Szpakowicz,S. "Logic Grammars" Byte Aug.1987. pp.185-195.

[TROS-88] Trostmann,E. e B.Palstrom. "CAD-based robot planning and control". IFAC Robot Control. pp. 465-469. FRG 1988

[WANG-89] Wang, K. e T.K.Lien. "The planning os a straight line trajectory via interactive computer graphics". Robotics and Computer-Integrated Manufacturing. V5 n2/3 pp. 215-221. 1989.

[WECK-87] Weck,M., Th.Niehaus e M.Osterwinter, "An interactive model based robot programming and simulation workstation" em "Off-line programming of Industrial Robots". Storr,A. e J.F.McWaters (Ed.). pp. 41-51. 1987

[WOZN-88] Wozniak, P. e J.Warczynski "Robot Simulation and Programming System". IFAC Robot Control pp. 437-442. FRG 1988